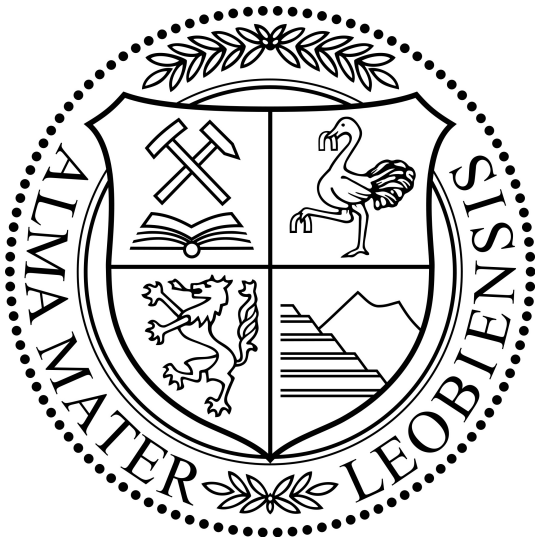
6 DOF Motion Control of a Robot-Driven Camera and its Application in Virtual Scenes



Diploma Thesis

Yuan Li

Supervisors
O.Univ.-Prof. Dipl.-Ing. Dr.techn. Paul O'Leary
Ass.Prof. Dipl.-Ing. Dr.mont. Gerhard Rath

March 2011

University of Leoben
Institute for Automation

EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre an Eides statt, dass ich diese Arbeit selbständig verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und mich auch sonst keiner unerlaubten Hilfsmittel bedient habe.

AFFIDAVIT

I declare in lieu of oath, that I wrote this thesis and performed the associated research myself, using only literature cited in this volume.

01.03.2011	Li Yuan
Datum / Date	Unterschrift / Signature

Acknowledgements

This thesis and my education would not have been possible without the help of many people.

First and foremost, I would like to express my thanks to Prof. Paul O'Leary and Gehard Rath, my supervisors, for all their time, help in English and German language and corrections.

Furthermore Gerold Probst for his support with software.

Doris Widek, the secretary of the institute for automation, for her help and encouragement.

My Mom and Dad, I want to thank for their endless love and full support.

Also my girl friend, Aijing Zhang, for her love and encouragement.

Finally, I would like to thank all my friends, near and far, for their encouragement and support.

Abstract

Nowadays cameras are used in many areas, for example image inspection and video games. This thesis explores a mathematical analysis for the interactive motion of a camera, which is moved by a robot and is controlled by a six degrees of freedom input device.

Motion of camera includes roll, pitch and yaw rotations and the translation in its own and the global coordinate systems. Furthermore, the inverse kinematics of the robot was used to find the angles of the drive motors to accomplish the desired motion. Denavit-Hartenberg convention was applied to model a standard six degrees of freedom industrial robot. Camera and robot are implemented and animated in a virtual world written in VRML code (Virtual Reality Modeling Language), which was generated with a CAD program. The algorithms were programmed in Matlab/Simulink, which provides interfaces to the 3D mouse for input control and to the virtual world for the purpose of animation.Introductions to the mathematics of VRML transforms and to robot kinematics were given.

After implementing and testing the equipment in a virtual world in VRML, the camera can be moved under the control of a 3D mouse. Now the algorithms are ready to be applied in a system with a real robot.

Zusamenfassung

Bewegte Kameras werden auf vielen Gebieten verwendet, zum Beispiel für Videoinspektion oder in Computerspielen. Ziel dieser Arbeit war die Erarbeitung der mathematischen Algorithmen für die Bewegung einer Kamera mit Hilfe eines Roboters mit sechs Freiheitsgraden, interaktiv gesteuert über eine 3D-Maus.

Die Bewegung umfasste die Rotation als Roll-, Nick-, und Gierwinkel, sowie die Translation im kameraeigenen und im globalen Koordinatensystem. Weiters war die inverse Kinematik für den Roboter erforderlich, um für die gewünschte Bewegung die Winkel der Gelenksantriebe zu ermitteln. Um den Roboter mit sechs Freiheitsgraden zu beschreiben, wurde die Denavit-Hartenberg Konvention verwendet. Kamera und Roboter wurden in einer virtuellen Welt implementiert, geschrieben in VRML (Virtual Reality Modeling Language). Der Code wurde mit einem CAD-Programm erzeugt. Die Algorithmen wurden programmiert in Matlab/Simulink, das auch Interfaces zur Verfügung stellte für die 3D-Maus als Eingabegerät und für die virtuelle Welt zum Zweck der Animation.

Eine Einführung in die Mathematik der VRML-Transformationen und in die Roboterkinematik wurde gegeben. Nach dem Implementieren und Testen kann die Kamera über die 3D-Maus gesteuert werden. Die Algorithmen können auf einem realen Kamera-Roboter-System eingesetzt werden.

Contents

1. Introduction

A camera is a device that records images. These images may be still photographs or moving images such as videos or movies [32]. Nowadays, cameras are used in many areas, video games, industrial image inspection, for example. Video games, Virtual worlds and interactive real time applications need a view of virtual world for human. For this purpose a virtual camera system must be used. To control the camera, we must study mathematical analysis of motion. Currently, motion is studied in different areas of research, simulation, game technology, animation, robotics, for example.

1.1. Motion in Games

Motion plays a very important role in computer games, and is also very important element for education and entertainment. Characters and objects are manipulated and moved subject to physical constraints. Motion in games include motion of camera, motion of virtual humans, etc. We have a lot of things to study about the motion, simulation of natural motion, navigation, physic based motion, algorithms and techniques of animation, manipulation of object, etc.

Motion of camera is a very important part of motion in games, in first person computer games, camera is the eyes of the player's character, in third person computer games, camera watches the player and moves to keep the player in sight. Nowadays we can play flight simulator games and navigate in a 3D world with a 3D mouse or with a joystick. In these games cameras moves through the scene like physical objects and make the experience realistic almost like in the real world. Like in a cinema movie, the camera connects us with the virtual world.

The mathematics of transformation of camera is very important for games and simulator. The basic mathematics for motion has been described in the literature [4], however, the detail of the mathematics for motion and control of camera with a six degrees of freedom device has not been studied well.

1.2. Motion of the Camera in VRML Worlds

VRML stands for Virtual Reality Modeling Language. The term VRML was coined by Dave Raggett in a paper submitted to The First International Conference on the World-Wide Web in 1994, and first discussed at the WWW94 VRML BOF established by Tim Berners-Lee, where Mark Pesce presented the Labyrinth demo he developed with Tony Parisi and Peter Kennard [32].

VRML is used to specify dynamic 3D scenes and the user can navigate with help of VRML browser. VRML scenes can be distributed over the World-Wide Web and browsed with VRML

browsers. In 1997, a new version of the format was finalized, as VRML97 also known as VRML 2.0, and became an ISO standard (ISO/IEC 14772-1:1997), this version is functional up to now [32]. We will use this version to study the motion of camera in VRML scenes. X3D is the succeeding standard, but VRML is still applied to education, electronic commerce, interactive system of entertainment, etc. VRML is very promising in those areas. Tody, CAD and CAM are very important in the mechanical and architectonic area, and VRML is used as an interchange standard for engineering data. A very important application of VRML is in the area of entertainment, VRML makes the interaction between human and virtual world more and more realistic. This is the purpose, why we will study the mathematics of motion in virtual worlds in VRML.

VRML world has no limits in size and complexity, this means a VRML world can be fairly large. A virtual world allows the user to move through it and to visit its parts. Nowadays the navigation of programs often is controlled by human with a 3D mouse. 3D mouse has more advantage than normal mouse for navigation. The basic mathematics for motion of camera have been described in literature [15] and [25], however, the mathematics for the continuous motion of a camera, which is controlled by a 3D mouse, has not been studied.

1.3. Motion of the Robot

George Devol applied for the first robotics patents in 1954 (granted in 1961). The first company to produce a robot was Unimation, founded by George Devol and Joseph F. Engelberger in 1956, and was based on Devol's original patents. An industrial robot is officially defined by ISO Standard 8373 as an automatically controlled, reprogrammable, multipurpose manipulator programmable in three or more axes. [32]

Two axes can reach any point in a plane, three axes can reach any point in space. If we want to fully control the orientation of the end effector three more axes are required.

Nowadays industrial robot is used in many area. Typical applications of robots are welding, painting, assembly, packaging and palletizing, product inspection, testing, all accomplished with high endurance, speed and precision. An important kind of inspection of products is image inspection, many kinds of cameras are used to industrial image inspection, for example infrared (IR) camera or video camera. For the purpose of camera motion with a robot, we need to study the motion of robot, this means we must study kinematics.

In robot kinematic analysis the position, velocity and acceleration of all links of the robot are calculated without considering the forces that cause this motion. The most active areas within robot kinematics is the screw theory. Robot kinematics are classified in two types, forward kinematics and inverse kinematics. The robot kinematics have been described in literature [11] and [29]. We need to study the motion of a robot, which is controlled by a 3D mouse, with a camera mounted on its end-effector.

The robot (see Fig. 1.1) we use is a kind of industrial robot. This robot has six rotation axes $(g_1 ... g_6)$ and each axis can rotate about 360°. Axis g_5 can rotate about $180°$.

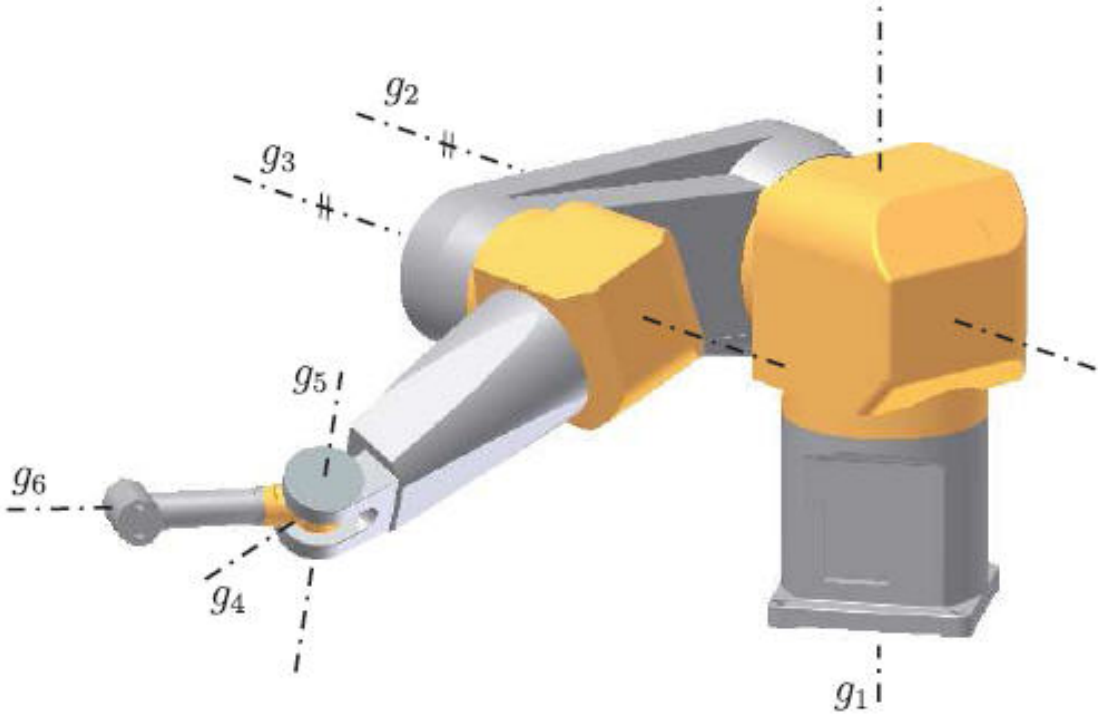


Figure 1.1.:Rotatation axes of the Stäubli-Robot [11].

2. Mathematics of VRML-Transforms

A virtual world is based on a 3D coordinate system (global coordinate system) and described by the Virtual Reality Modeling Language (VRML), which is used for representing 3D interactive vector graphics. Every object has its own coordinate system, the position of every object is described in the global coordinate system and in its own coordinate system. We can change position of every object in VRML world. Position change of objects in the real world is called motion, but motion of object in VRML world is called a transform. Transformation includes rotation, translation and scaling, etc. Transformation is based on mathematics of motion. In this chapter we will study mathematics of motion.

2.1. Mathematical Notation

x, y, z	Initially coordinate
x', y', z'	Coordinate after transform
a, b, c	Axial displacement
θx, θy, θz	Rotate angle about axis (Euler angles)
θ	Rotate angle about axis $\vec{u}$ = (u1,u2,u3)
$\vec{u}$	Unit vector
$\vec{a}$	Vector
I	Identity matix
Rx, Ry, Rz	Rotate matrix about axis
V'	Matrix after transform
V	Initially matrix
S	Scaling matrix
Sx, Sy, Sz	Scaling factors
α	is the angle between the x-axis and the line of nodes (euler angle)
β	is the angle between the z-axis and the Z-axis (euler angle)
γ	is the angle between the line of nodes and the X-axis (euler angle)
q	Unit quaternion
s	Scalar
i, j, k	Imaginary part
Qx, Qy, Qz	Quaternion
q_{rot}	Rotated quaternion

2.2. Coordinate System

The VRML uses right-handed Cartesian coordinate system and is different from the Matlab (see Fig. 2.1). VRML uses the world coordinate system in which the y-axis points upward and the

z-axis places objects nearer or farther from the front of the screen [22]. It is very important to understand this fact in the situation, which includes interaction of these different coordinate systems.

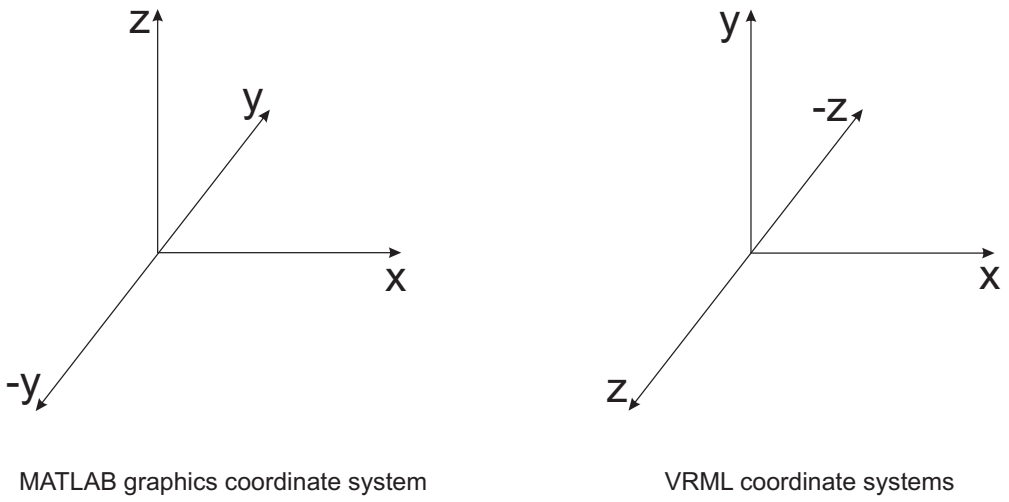


Figure 2.1.: Coordinate Systems

2.3. Translation

In Euclidean geometry, a translation is moving every point a constant distance in a specified direction [32]. Translation can be described as addition of a constant vector to each point of an object. This refers to move objects to different places in the coordinate system. To translate a 3D point, modify each dimension separately [4]:

$$x' = x + a; y' = y + b; z' = z + c; \tag{2.1}$$

In matrix format of homogeneous coordinates [11]:

$$\begin{bmatrix} 1 \\ x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ a & 1 & 0 & 0 \\ b & 0 & 1 & 0 \\ c & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ x \\ y \\ z \end{bmatrix} \tag{2.2}$$

The code for translation with VRML:

```
Transform{   a   b   c
translation     0.0 0.0 0.0
children [ . . . ]
}
```

To describe the problem between different coordinate system, we builded a coordinate system in VRML world with CINEMA 4D[1] (see Fig. 2.2):

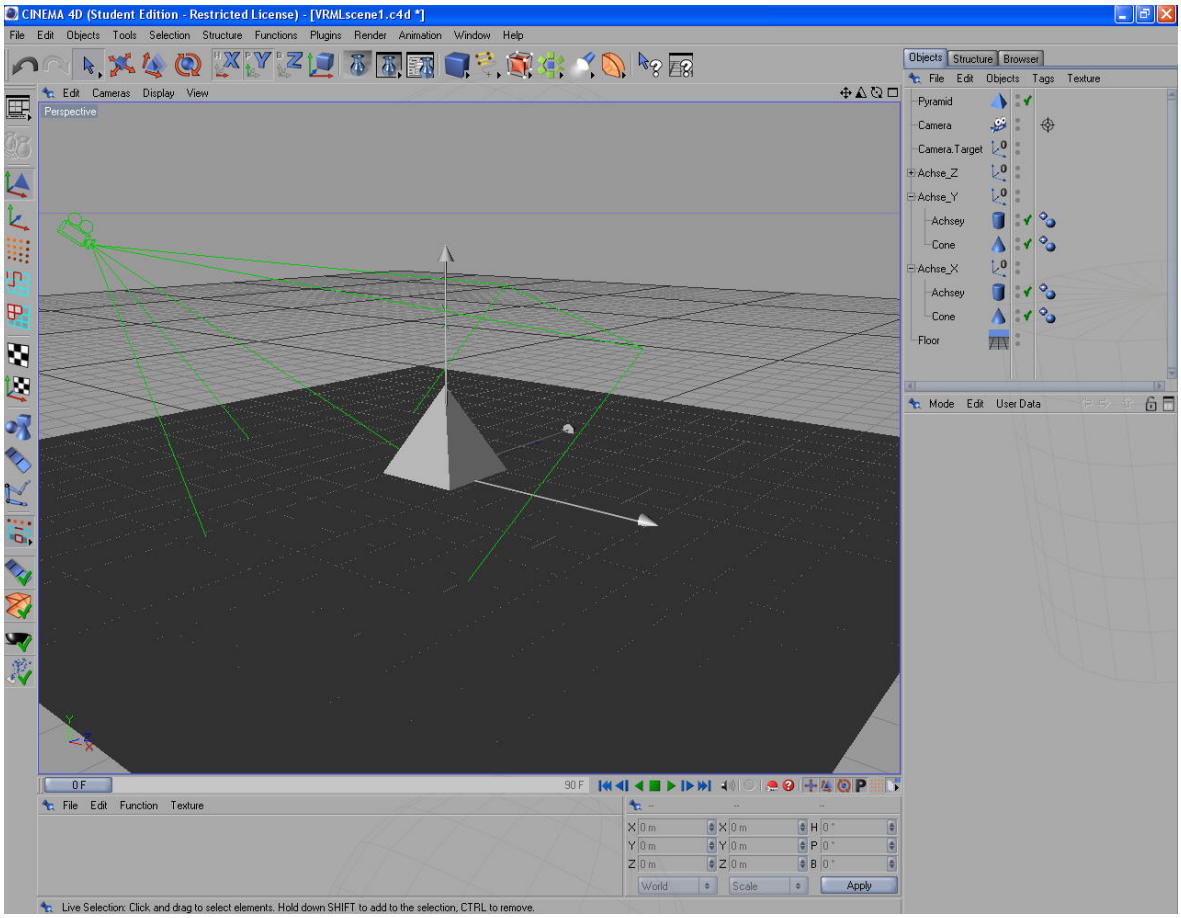


Figure 2.2.: Building an object in a coordinate system of a virtual world using CINEMA 4D

Then we developed the code of simulation with Matlab/Simulink. First of all we used a VR Sink to load a VRML file and choose ports which write values of transformation in VRML world (see Fig. 2.3). In Matlab/Simulink, we can build a connection between Matlab and VRML world to define the behavior of VRML object. This means All parameters of VRML world, which are used to control the motion in VRML world, could be controlled by Matlab/Simulink.

[1] Cinema 4D is a software to create virtual movie scenes and 3D animations. It is capable of exporting a scene to a VRML file.

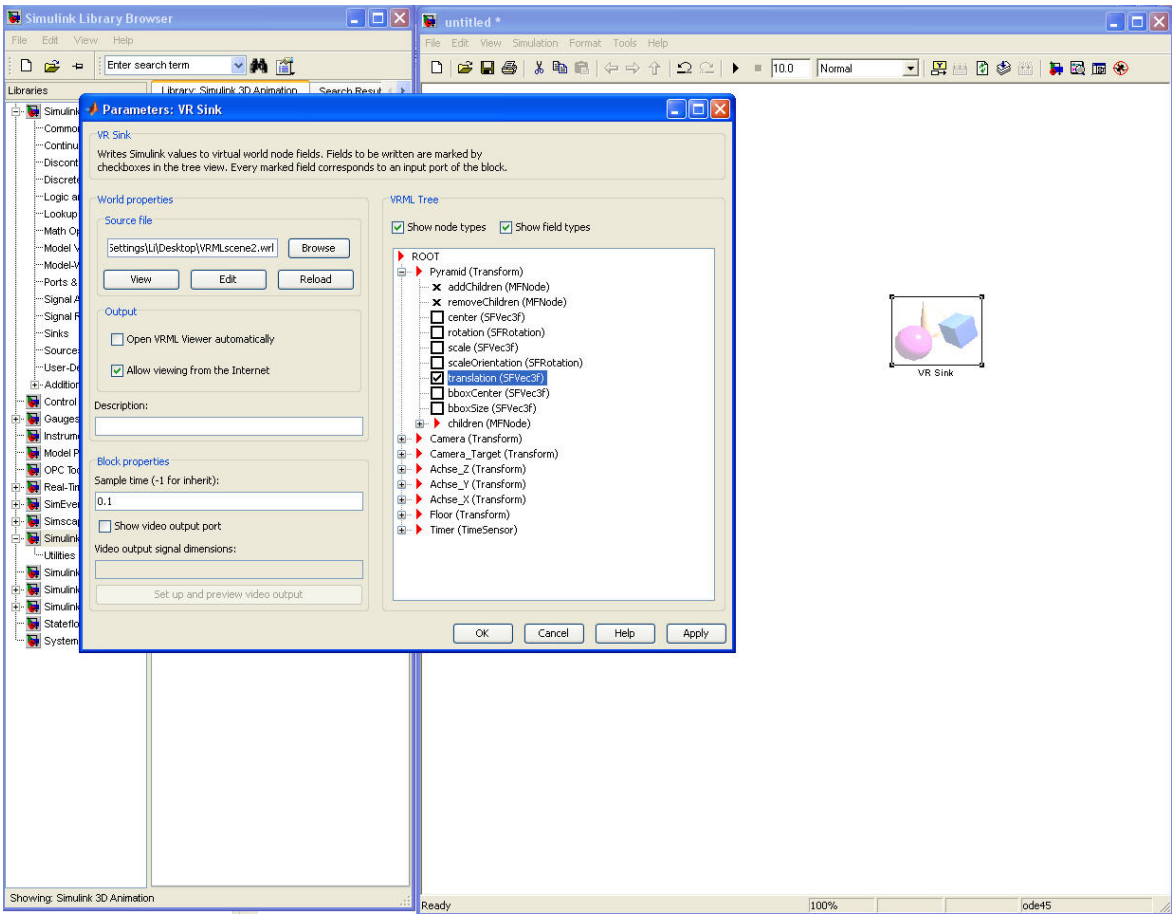


Figure 2.3.: Using VR sink to load a VRML file

To move an object over time, the value of coordinate of VRML object is controlled using the ramp[1] (see Fig. 2.4).

We must change the initial value of Ramp to a three-dimensional value (see Fig. 2.5), which is used to move an object in 3D world.

When we changed the first value, the prism goes along z axis of the global coordinate system, this means z-value of prism increase over time.
When we changed the second value, the prism goes along y axis of the global coordinate system, this means y-value of prism increase over time.
When we changed the third value, the prism goes along x axis of the global coordinate system, this means x-value of prism increase over time.

[1]The Ramp block generates a signal that starts at a specified time and value and changes by a specified rate. The block's Slope, Start time, and Initial output parameters determine the characteristics of the output signal. All must have the same dimensions after scalar expansion [22].

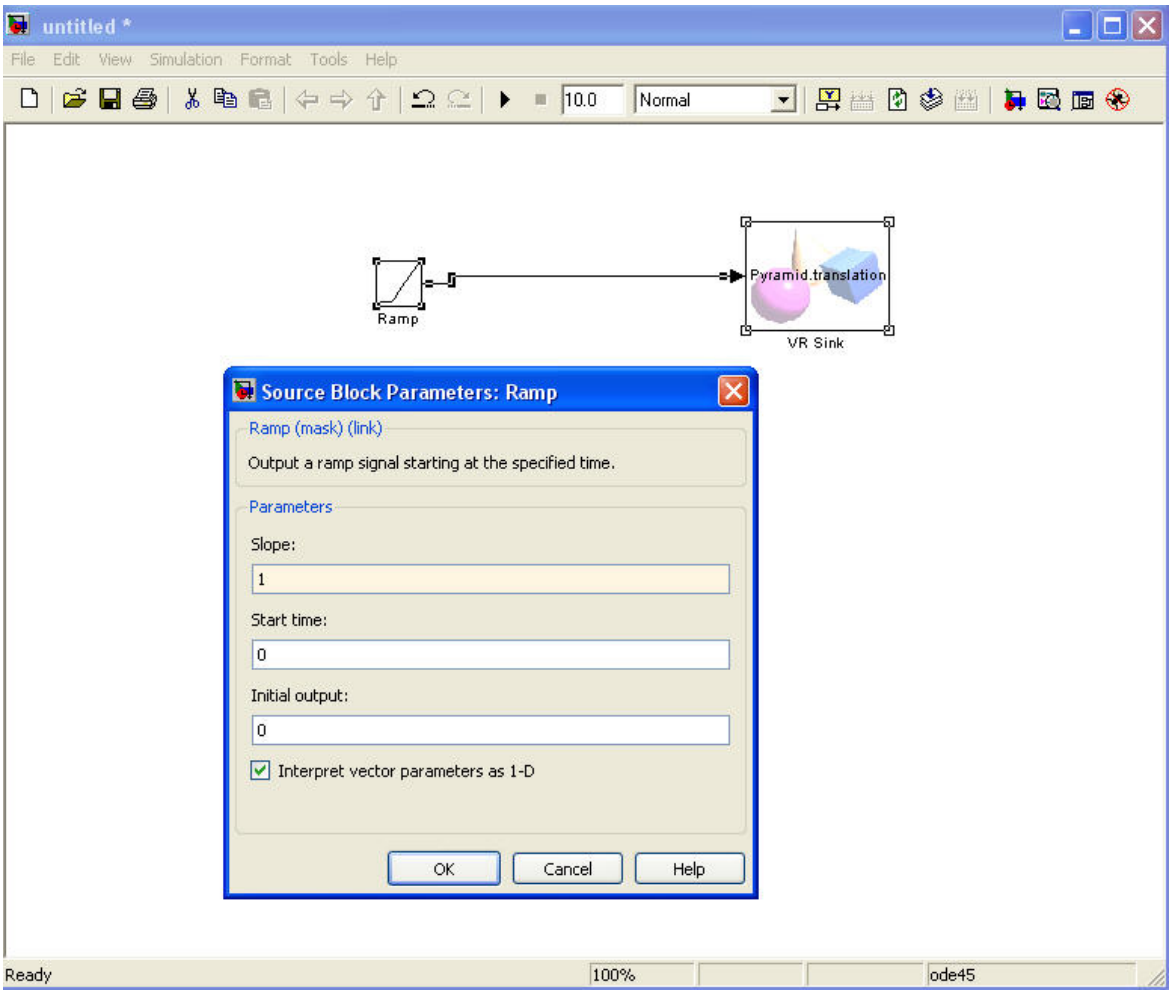


Figure 2.4.: Develop the code of simulation using Matlab/Simulink

2.4. Rotation

A rotation is a circular movement of an object around a center (or point) of rotation. A three-dimensional object rotates always around an imaginary line called a rotation axis. Mathematically, a rotation is a rigid body[1] movement, which is unlike translation, keeps a point fixed. This definition applies to rotations within both two and three dimensions (in a plane and in space, respectively) [32]. In three-dimensional space a rotation keeps a line fixed, which is represented by a vector, which characterizes the rotation at that point during its attributes (length and direction). This follows from Euler's rotation theorem. In flight dynamics, the principal rotations are known as pitch, roll and yaw [32]. Rotation about the x-axis is often called Roll, Rotation about the y-axis is often called Pitch, Rotation about the z-axis is often called Yaw. Then we must know the convention of rotation, rotate angle is positive, if the rotation in counterclockwise, which is determined by the righthand rule.

[1] In physics, a rigid body is an idealization of a solid body of finite size in which deformation is neglected. In other words, the distance between any two given points of a rigid body remains constant in time regardless of external forces exerted on it [32].

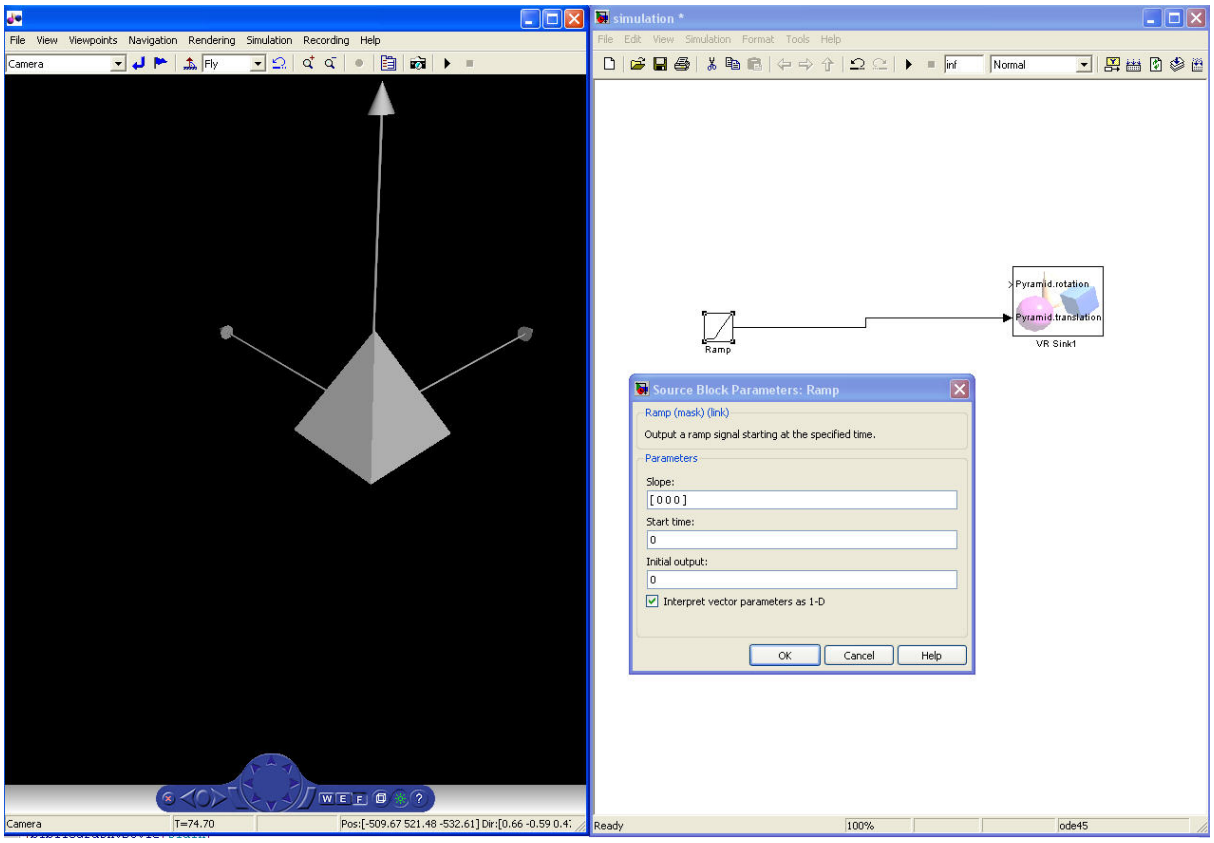


Figure 2.5.: Using Ramp to give a value of coordinate system

A 3D object rotate about x-axis is described in Fig. 2.6,

Modify each dimension separately [4]:

$$y' = y\cos\theta x - z\sin\theta x; z' = y\sin\theta x + z\cos\theta x; x' = x \tag{2.3}$$

In Matrix format [4]:

$$V' = \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = R_x \cdot V = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta_x) & -\sin(\theta_x) \\ 0 & \sin(\theta_x) & \cos(\theta_x) \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{2.4}$$

Rotation of 3D object about y-axis is described in Fig. 2.7,

Modify each dimension separately [4]:

$$z' = z\cos\theta y - x\sin\theta y; x' = z\sin\theta y + x\cos\theta y; y' = y \tag{2.5}$$

In matrix format [4]:

$$V' = \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = R_y \cdot V = \begin{bmatrix} \cos(\theta_y) & 0 & \sin(\theta_y) \\ 0 & 1 & 0 \\ -\sin(\theta_y) & 0 & \cos(\theta_y) \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{2.6}$$

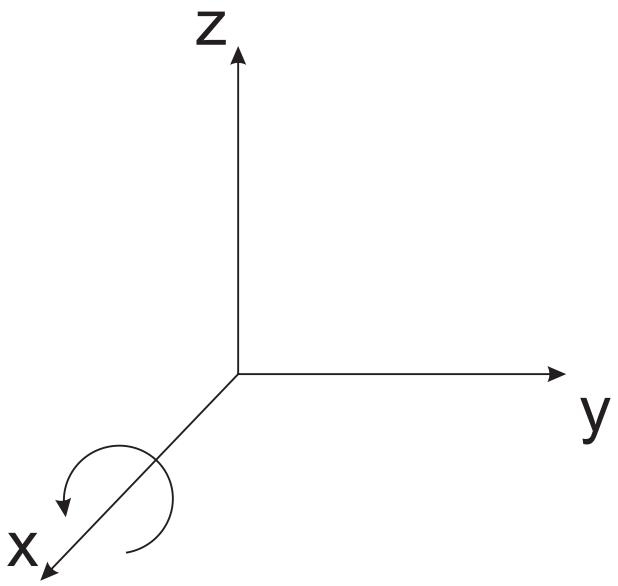


Figure 2.6.: Rotation about x-axis

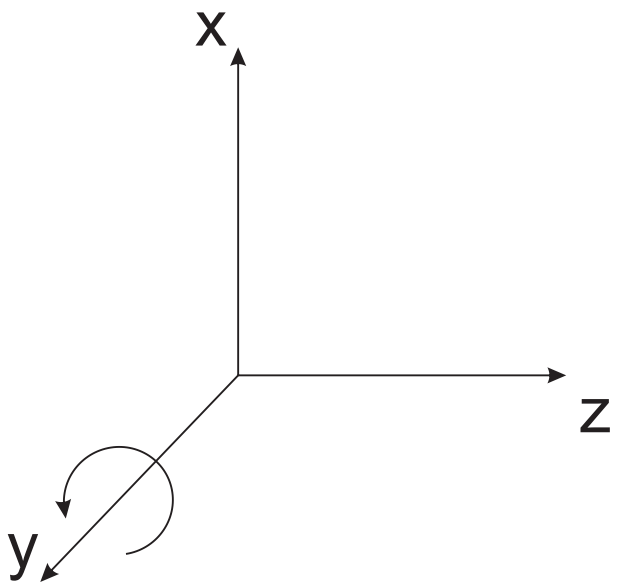


Figure 2.7.: Rotation about y-axis

Then rotation of a 3D object about z-axis (see Fig. 2.8), modify each dimension separately [4]:

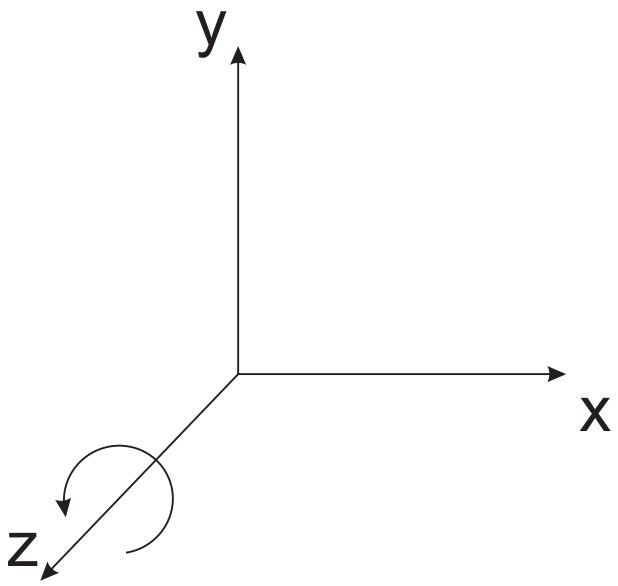


Figure 2.8.: Rotation about z-axis

$$x' = x\cos\theta z - y\sin\theta z; y' = x\sin\theta z + y\cos\theta z; z' = z \tag{2.7}$$

In matrix format [4]:

$$V' = \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = R_z \cdot V = \begin{bmatrix} \cos(\theta_z) & -\sin(\theta_z) & 0 \\ \sin(\theta_z) & \cos(\theta_z) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{2.8}$$

Then we must know the VRML code for rotation:

Transform{ x y z angle
rotation 0.0 0.0 0.0 0.0
children [. . .]
}

From VRML code of rotation, we know that, if we want to rotate an object in VRML world, we need four parameters as following equation:

$$\begin{bmatrix} a & b & c & d \end{bmatrix} \tag{2.9}$$

Where, a, b, c are direction vectors of the rotation axis, d is the rotation angle about rotation axis. Then we changed the initial value of Ramp (see Fig. 2.9):

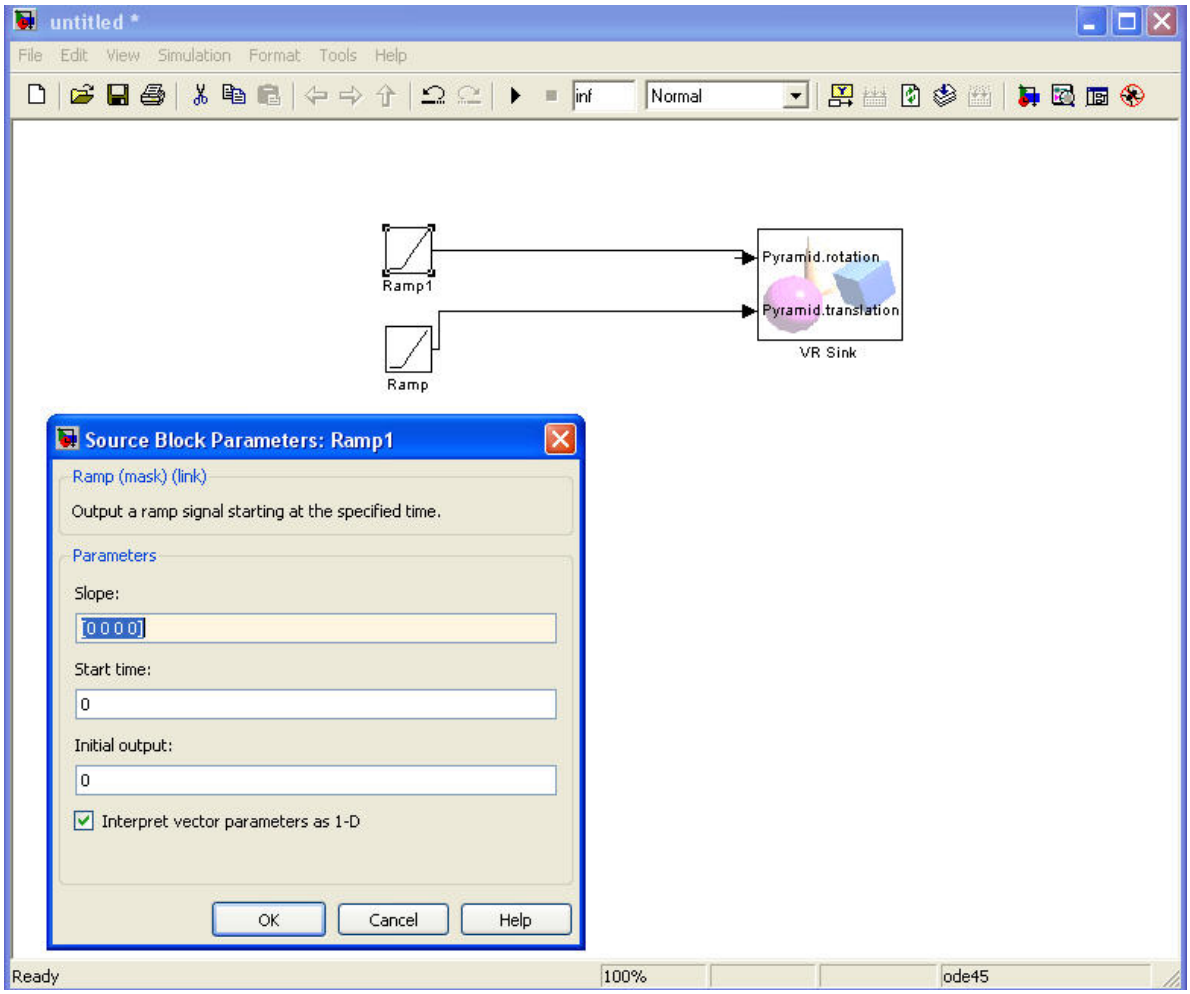


Figure 2.9.: Code of Ramp for rotating an object in VRML world

We gave the fourth parameter a positive value for each rotation and when we gave the first parameter a positive value, the prism rotates about z-axis counterclockwise, when we gave the second parameter a positive value, the prism rotates about y-axis counterclockwise, when we gave a third parameter a positive value, the prism rotates about x-axis counterclockwise. This is the same as the convention.

If we need more than one rotation of object about different axes, we can multiply each of respective rotation matrices in the order of applied rotation. For example, if the object rotates first about x-axis, then rotates about y-axis, at last rotates about z-axis are applied, the combined rotation is [4]:

$$R = Rx \cdot Ry \cdot Rz \tag{2.10}$$

Then the rotation matrix is as following equation:

$$R = \begin{bmatrix} \cos\theta y \cos\theta x & -\cos\theta y \sin\theta z & -\sin\theta y \\ -\sin\theta x \sin\theta y \cos\theta z + \cos\theta x \sin\theta z & \sin\theta x \sin\theta y \sin\theta z + \cos\theta x \cos\theta z & -\sin\theta x \cos\theta y \\ \cos\theta x \sin\theta y \cos\theta z + \sin\theta x \sin\theta z & -\cos\theta x \sin\theta y \sin\theta z + \sin\theta x \cos\theta z & \cos\theta x \cos\theta y \end{bmatrix} \tag{2.11}$$

If rotation about z-axis is applied first, then rotation about x-axis followed by rotation about y-axis, the combined rotation matrix is:

$$R = Rz \cdot Rx \cdot Ry \tag{2.12}$$

Then we look at axis of rotation. Every rotation in three dimensions has an axis, which has direction that is fixed by the rotation. Given a rotation matrix R, a vector $\vec{u}$ parallel to the rotation axis must satisfy:

$$R\vec{u} = \vec{u} \tag{2.13}$$

Rotation matrix gives an axis and an angle, the matrix for a rotation by an angle of θ about an axis the direction of $\vec{u}$ is [11]:

$$R = \begin{bmatrix} u_x^2 + (1 - u_x^2)\cos\theta & u_x u_y(1 - \cos\theta) - u_z \sin\theta & u_x u_z(1 - \cos\theta) + u_y s \\ u_x u_y(1 - \cos\theta) + u_z \sin\theta & u_y^2 + (1 - u_y^2)\cos\theta & u_y u_z(1 - \cos\theta) - u_x s \\ u_x u_z(1 - \cos\theta) - u_y \sin\theta & u_y u_z(1 - \cos\theta) + u_x \sin\theta & u_z^2 + (1 - u_z^2)\cos\theta \end{bmatrix} \tag{2.14}$$

Where:

$$\vec{u} = \begin{bmatrix} u_x \\ u_y \\ u_z \end{bmatrix} \tag{2.15}$$

This can be written as Rodrigues' formula [11]:

$$R_{u,\theta} = uu^T + \cos\theta(\mathrm{I} - uu^T) + [u]_\times \sin\theta \tag{2.16}$$

Equivalent to:

$$R_{u,\theta} = \mathrm{I} + [u]_\times \sin\theta + [u]_\times^2 (1 - \cos\theta) \tag{2.17}$$

Consider the cross product as a matrix multiplication:

$$u \times v = [u]_\times v \tag{2.18}$$

Where:

$$[u]_\times = \begin{bmatrix} 0 & -u_z & u_y \\ u_z & 0 & -u_x \\ -u_y & u_x & 0 \end{bmatrix} \tag{2.19}$$

$[u]_\times$ is the skew symmetric form of $\vec{u}$, Then:

$$uu^T = \begin{bmatrix} u_x \\ u_y \\ u_z \end{bmatrix} \cdot \begin{bmatrix} u_x & u_y & u_z \end{bmatrix} = \begin{bmatrix} u_x^2 & u_x u_y & u_x u_z \\ u_x u_y & u_y^2 & u_y u_z \\ u_x u_z & u_y u_z & u_z^2 \end{bmatrix} \tag{2.20}$$

One way to represent a rotation in 3D are Euler angles (see Fig. 2.10), Euler angles are a means of representing the spatial orientation of any coordinate system as a composition of rotations from a coordinate system. In the following the fixed system is described in lower case (x, y, z) and the rotated system is described in upper case letters (X, Y, Z)[32]. Euler angles α, β, γ are composition of Euler rotations.

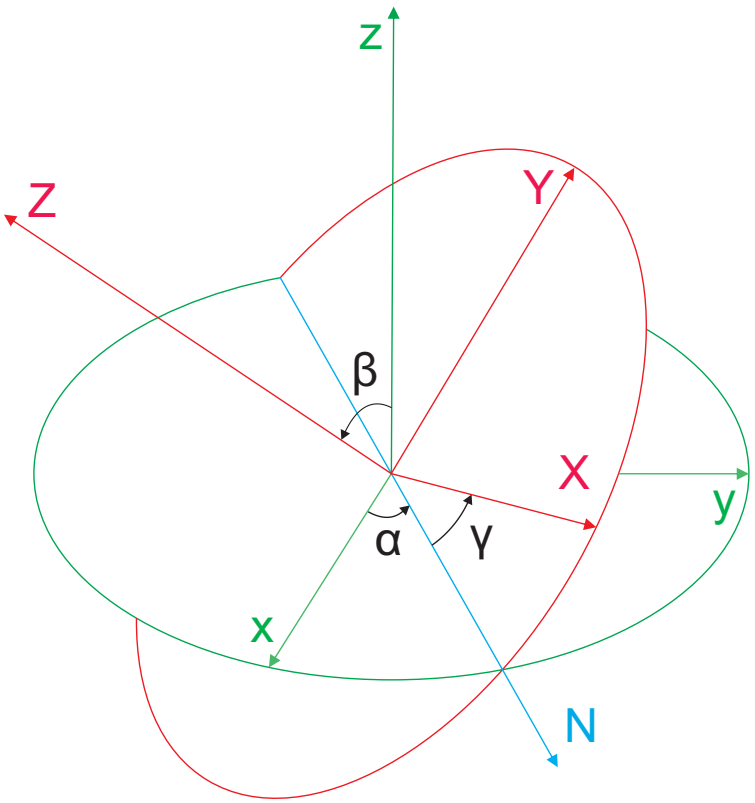


Figure 2.10.: Euler angles: The x-y-z (fixed) system is shown in green, the X-Y-Z (rotated) system is shown in red. The line of nodes, labeled N, is shown in blue.

If we want to define Euler angles, we must give a reference frame and the one whose orientation we want to describe, first we define the line of nodes (N) as the intersection of the $x - y$ and the $X - Y$ coordinate planes [32]. Then we can define Euler angles as (see Fig. 2.10):

- α is the angle between the x-axis and the line of nodes.
- β is the angle between the z-axis and the Z-axis.
- γ is the angle between the line of nodes and the X-axis.

The problem of Euler rotation is the gimbal lock (see Fig. 2.11), which is the loss of one degree of freedom that occurs when the axes of two of the three gimbals[1] are driven into the same place and cannot compensate for rotations around one axis in three dimensional space [32]. Euler angles give a numerical description of any rotation in three dimensional space using three numbers. To make a comparison, all the translations can be described using three numbers x, y and z, as the succession of three consecutive linear movements along three perpendicular axes X, Y and Z. That is the same for rotations, all the rotations can be described using three numbers α β and γ as the succession of three rotational movements around three axes that are perpendicular one to the next. This similarity between linear coordinates and angular coordinates makes Euler angles very intuitive, but unfortunately they suffer from the gimbal lock problem [32].

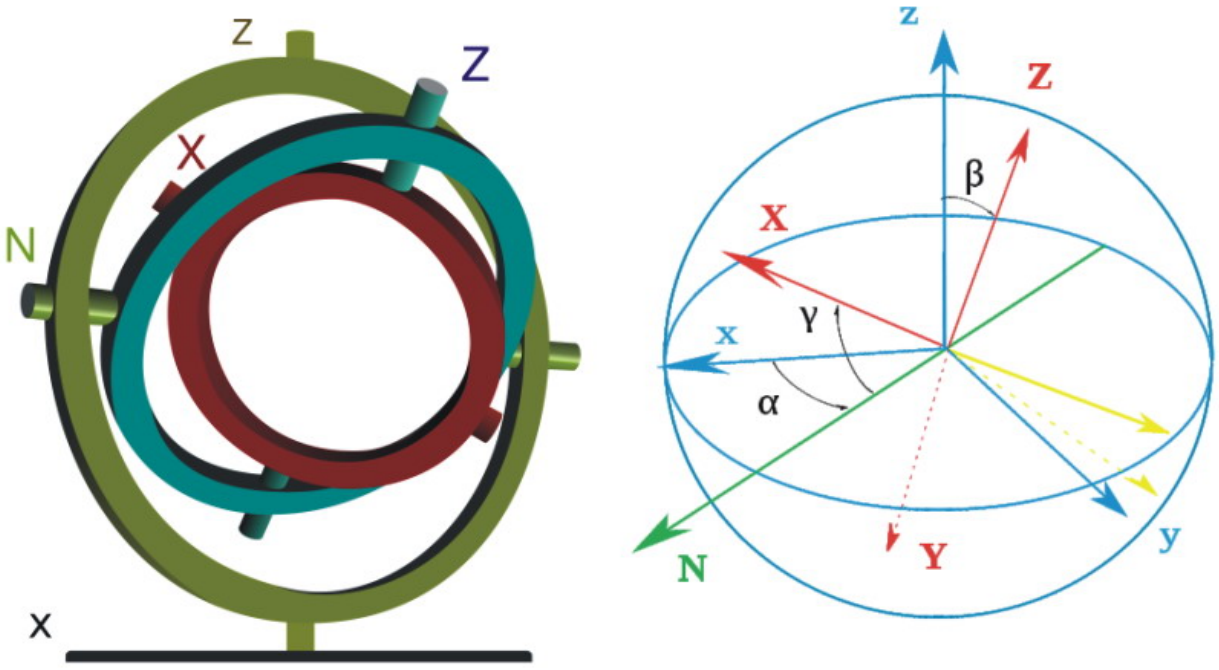


Figure 2.11.: Three axes Z-X-z-gimbal showing Euler angles. External frame and external axis 'x' are not shown. Axes 'Y' and 'y' are perpendicular to each gimbal ring [32].

To explain this problem, we look at a rotation in 3D space, which can be represented with matrices:

$$R = \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\beta & -\sin\beta \\ 0 & \sin\beta & \cos\beta \end{bmatrix} \begin{bmatrix} \cos\gamma & -\sin\gamma & 0 \\ \sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.21}$$

with α and γ are constrained in the interval $[-\pi, \pi]$, and β constrained in the interval $[0, \pi]$. For example, if $\beta = 0$, then $\sin\beta = 0$ and $\cos\beta = 1$, the above expression becomes equal to:

[0]A gimbal is a ring that is suspended so it can rotate about an axis. Gimbals are typically nested one within another to accommodate rotation about multiple axes [32].

$$R = \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\gamma & -\sin\gamma & 0 \\ \sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.22}$$

The second matrix is the identity matrix and has no effect on the product. Carrying out matrix multiplication of first and third matrices:

$$R = \begin{bmatrix} \cos\alpha\cos\gamma - \sin\alpha\sin\gamma & -\cos\alpha\sin\gamma - \sin\alpha\cos\gamma & 0 \\ \cos\alpha\cos\gamma + \cos\alpha\sin\gamma & -\sin\alpha\sin\gamma + \cos\alpha\cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.23}$$

And finally using the trigonometry formulas:

$$R = \begin{bmatrix} \cos(\alpha+\beta) & -\sin(\alpha+\beta) & 0 \\ \sin(\alpha+\beta) & \cos(\alpha+\beta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.24}$$

Changing value of α and β in the above matrix has the same effects, the rotation angle $\alpha + \beta$ changes, but the rotation axis remains in the Z direction. The last column and the last line in the matrix won't change: one degree of freedom has been lost.

The only solution for α and β to recover different roles is to get β away from the zero value.

A similar problem appears when $\beta = \pi$.

One can choose another convention for representing a rotation with a matrix using the Euler angles than the $Z - X - z$ convention, and also choose other variation intervals for the angles, but at the end there is always at least one value for which a degree of freedom is lost.

One can use quaternion to solution this problem. Quaternion is another representation for rotations in 3D space. In mathematics, the quaternions are a number system that extends the complex numbers. They were first described by Irish mathematician Sir William Rowan Hamilton in 1843 and applied to mechanics in three-dimensional space [32]. There are many ways to represent the orientation of an object. Most programmers use 3×3 rotation matrices or three Euler angles to store this information. Each of these solutions works fine until you try to interpolate smoothly between two orientations of an object. A quaternion is a tuple made of four numbers [13]:

$$\begin{array}{c} q = s + ix + jy + kz \\ s ... Scalar\ part \\ x, y, z ... Vector\ part \end{array} \tag{2.25}$$

or,

$$q = \left[s, \begin{pmatrix} x \\ y \\ z \end{pmatrix} \right] \tag{2.26}$$

Conjugation of a quaternion is [13]:

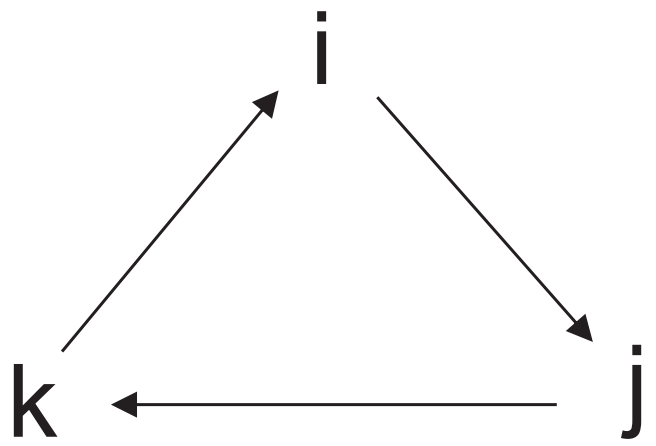


Figure 2.12.:Calculation rule for the multiplication of imaginary part

$$q^* = \left[s, -\begin{pmatrix} x \\ y \\ z \end{pmatrix} \right] \tag{2.27}$$

Reciprocal of a quaternion is [13]:

$$\mathrm{q}^{-1} = \frac{q^*}{qq^*} \tag{2.28}$$

The magnitude of a quaternion is [13]:

$$\|q\| = Norm(q) = \sqrt{qq^*} = \sqrt{s^2 + x^2 + y^2 + z^2} \tag{2.29}$$

If the relation $s^2 + x^2 + y^2 + z^2 = 1$ is verified, then the quaternion is a unit quaternion and can be used to represent a rotation. If a quaternion is a unit quaternion $q^* = q^{-1}$. To change a quaternion to a rotation around an arbitrary axis in 3D space,we should take a rotation angle θ about axis $\vec{u} = (u1, u2, u3)$, $\vec{u}$ is a unit vector, then we get the quaternion $q = (w, x, y, z)$ like [13]:

$$\begin{aligned} w &= \cos(\theta/2); \\ x &= u1 \sin(\theta/2); \\ y &= u2 \sin(\theta/2); \\ z &= u3 \sin(\theta/2) \end{aligned} \tag{2.30}$$

$$\mathrm{q} = \|\mathrm{q}\| \cdot (\cos(\theta/2), \mathrm{u1} \cdot \sin(\theta/2), \mathrm{u2} \cdot \sin(\theta/2), \mathrm{u3} \cdot \sin(\theta/2)) \tag{2.31}$$

We want to rotate a vector $\vec{a} = (a1, a2, a3)$ about axis $\vec{u}$, then we need to translate $\vec{a}$ to quaternion $\vec{p} = (0, a1, a2, a3)$ [13]:

$$p_{rot} = q \cdot p \cdot q^{-1} \tag{2.32}$$

If we have three Euler angles (α, β, γ), then we can form three independent quaternions [13]:

$$\begin{array}{l} \mathrm{Qx} = [\cos(\alpha/2), (\sin(\alpha/2), 0, 0)]\,; \\ \mathrm{Qy} = [\cos(\beta/2), (0, \sin(\beta/2), 0)]\,; \\ \mathrm{Qz} = [\cos(\gamma/2), (0, 0, \sin(\gamma/2))] \end{array} \tag{2.33}$$

And the final quaternion is obtained by $Q_x \cdot Q_y \cdot Q_z$.

2.5. Scaling

In Euclidean geometry, uniform scaling is a linear transformation that enlarges or increases or diminishes objects by a scale factor the scale factor that is the same in all directions, it is also called a homothety [32]. The result of uniform scaling is similar to the original. A scale factor of one is normally allowed, so that congruent shapes are also classified as similar, but some school text books specifically exclude this possibility.

More general is scaling with a separate scale factor for each axis direction. Non-uniform or anisotropic scaling is obtained when at least one of the scaling factors is different from the others, a special case is directional scaling (in one direction). Non-uniform scaling changes the shape of the object, e.g. a rectangle may change into a rectangle of a different shape, but also into a parallelogram (the angles between lines parallel to the axes are preserved, but not all angles) [32].

Scaling in VRML-World means enlarge or shrink shapes, the mathematics of scaling is [27]:

$$V' = VS = \begin{bmatrix} x & y & z & 1 \end{bmatrix} \begin{bmatrix} Sx & 0 & 0 & 0 \\ 0 & Sy & 0 & 0 \\ 0 & 0 & Sz & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.34}$$

where Sx, Sy, Sz are the scale factors in the x, y, z directions.

3. Camera Motion in a Virtual World

In the previous chapter we studied mathematics of motion in 3D world. We already know how rotation and translation of an object in the global coordinate system are described by mathematic equations. Now we study the motion of a camera, which is built in VRML world and controlled by a 3D mouse. In our project we do not consider the gravitation and acceleration, which are very important and influence motion in real world. We only study mathematics of the motion of camera in ideal situation.

3.1. 3D Mouse

We used a six degrees of freedom device (see Fig. 3.1) in the experiment, which was used to control the position of camera in the virtual world. 3D Mouse is the perfect input device for building and manipulating 3D models and Virtual Reality Worlds, first of all we installed the driver of 3D mouse in our computer and connected this USB device to it.

Figure 3.1.:3D Mouse with six degrees of freedom

The outputs of this 3D mouse could be used in different ways (see Fig. 3.2): The movements

Figure 3.2.:Movement of the space mouse

"Roll", "Tilt" and "Spin" control the rotation about x, y and z axis, like "Roll", "Pitch" and "Yaw". The movement "Left/Right", "Up/Down" and "Forward/backward" control the translation about x, y and z axis.

3.2. Development of the Matlab Code

In Chapter 2, we described how to use Matlab/Simulink for the connection of the VRML world with the controller. In this section we describe how to develop a Matlab code for our experiment. At first we used the Matlab/Simulink 3D Animation and then chose the "Space mouse Input[1]" and "VR Sink" (see Fig. 3.3):

The Space mouse input block has three outputs, they are "Speed", "Position" and "Viewpoint Coordinates" (see Fig. 3.4):

"Speed" means that no transformations are done. Outputs are translation and rotation speeds [22].

"Position" means that translations and rotations are integrated. Outputs are position and orientation in the form of roll/pitch/yaw angles [22].

"Viewpoint coordinates" means that translations and rotations are integrated. Outputs are position and orientation in the form of an axis and an angle. You can use these values as viewpoint coordinates in VRML [22].

The type of output which we chose is "speed" for our experiment. We chose an "Integrator" witch from Matlab/Simulink-Commonly used blocks to integrate speed over time, this means the "speed" signal is changed into distance signal in VRML world. We also chose a "Gain" (from Matlab/Simulink-Commonly) to change the scale of the "Speed" signal (see Fig. 3.5):

3.3. VR Sink

To study the motion of camera in VRML world, we used the "VR Sink" block, which can write data from Matlab/Simulink model to virtual world. It loads a VRML file and one can choose the port for motion to write values from the 3D mouse into the VRML world (see Fig. 3.6):

[1] A space mouse is a device similar to a joystick in purpose, but it also provides movement control with six degrees of freedom. This block reads the status of the space mouse and provides some commonly used transformations of the input [22].

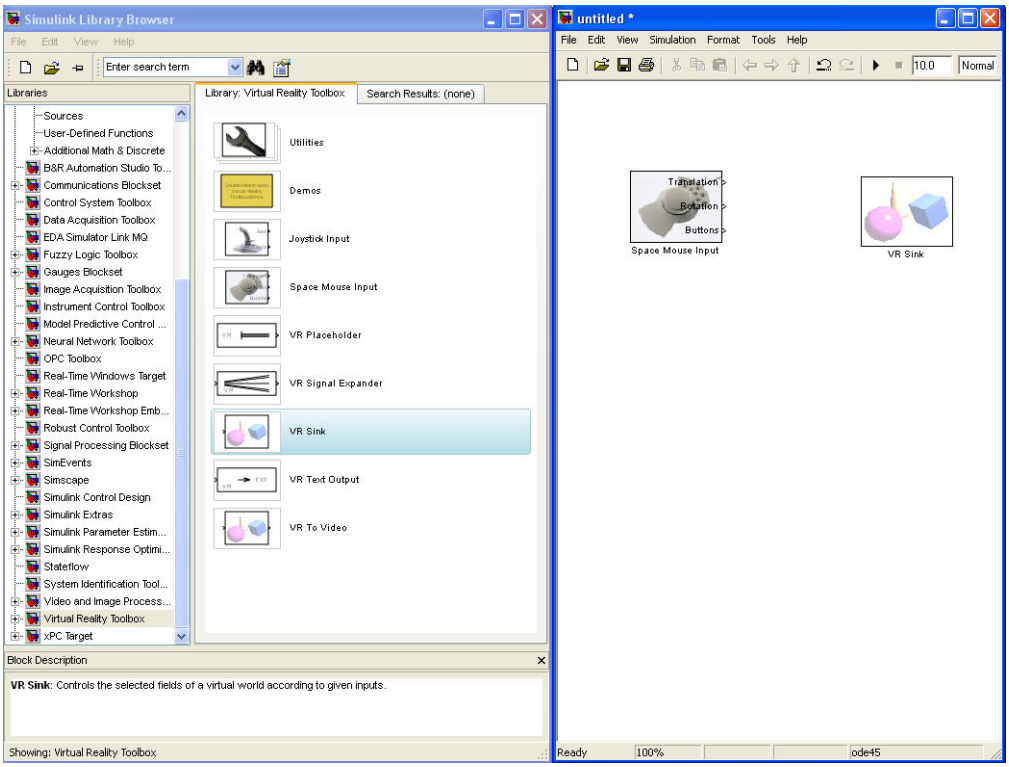


Figure 3.3.:Matlab/simulink blocks for 3D mouse and VRML-World

There are two types of translation and rotation for the camera (see Fig. 3.7).

The first type of transformation of the camera is rotation and translation of the camera in the global coordinate system of VRML world, this means we can control the axis-value of the coordinate system, direction and angle of rotation of camera in the global coordinate system when we give values to the ports "$Camera.Rotation$" and "$Camera.Translation$" with 3D mouse. The second type of the transformation of the camera is rotation and translation of the camera in its own coordinate system, which is based on the global coordinate system. This means we can write value of the coordinate system, direction and angle of rotation of camera in the own coordinate system after giving values to the ports "$View\_camera.position$" and "$View\_camera.Orientation$" with the 3D mouse.

After this, one can control the coordinates and directions of the camera and the objects with the 3D mouse. First we wanted to let the camera focus on the object when the position of the camera changed in global coordinate system by 3D mouse. We needed to develop a mathematical basis for this transformation, fist we needed to know the position of camera in VRML world (see Fig. 3.8), second, we needed to know how works the camera in VRML world. We saw that the camera focus along z-axis (Direction Vector is (0,0,1)) of the own coordinate system.

The input signal of the rotation of the camera must in the form "[Direction Vector, Rotation Angle]":

The "Direction Vector" between point O(0,0,0) and A(x1,y1,z1) is:

$$\vec{a} = \overrightarrow{OA} = (x1, y1, z1) - (0, 0, 0) = (x1, y1, z1) \tag{3.1}$$

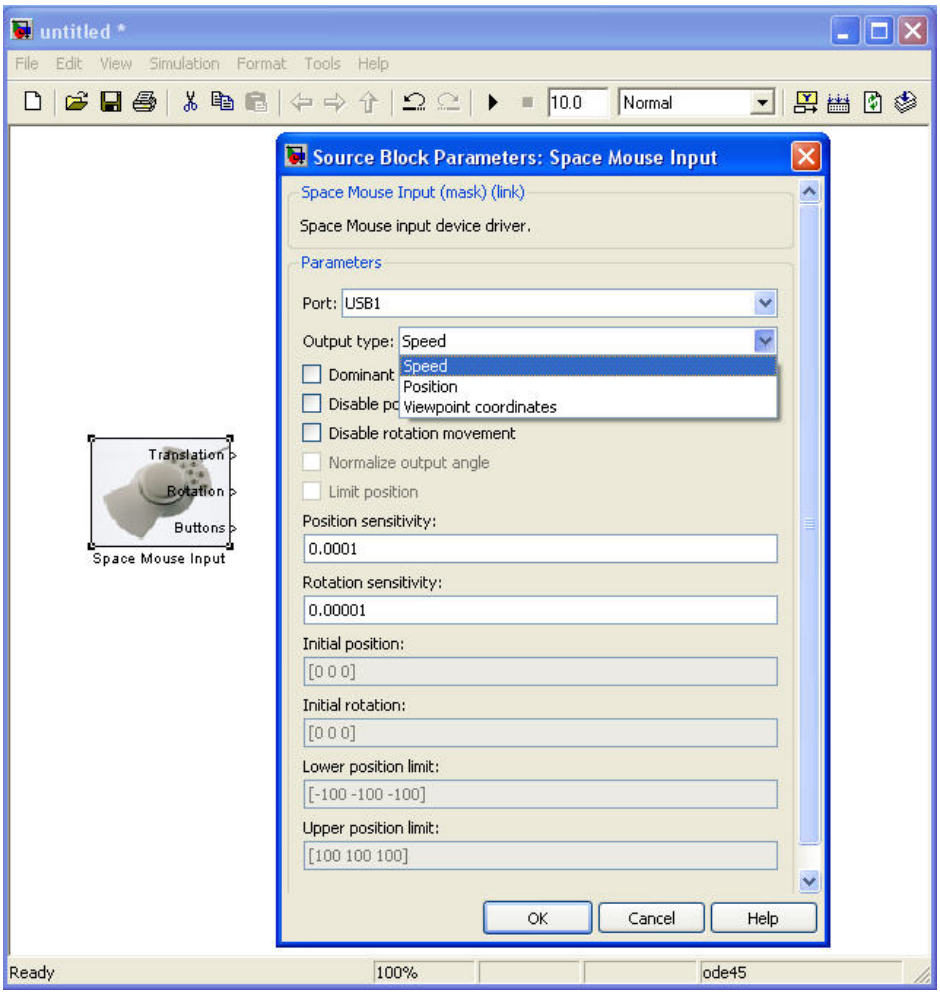


Figure 3.4.:Three outputs of Space mouse input block

The vector analysis yields (see Fig. 3.9):

$$\overrightarrow{OA} + \overrightarrow{AB} = \overrightarrow{OB}; \overrightarrow{OB} - \overrightarrow{OA} = \overrightarrow{AB} \tag{3.2}$$

This can be also written as:

$$\vec{a} + \vec{c} = \vec{b}; \vec{b} - \vec{a} = \vec{c} \tag{3.3}$$

The length of the vector a can be computed with the Euclidean norm:

$$\|\vec{a}\| = \sqrt{x1^2 + y1^2 + z1^2} \tag{3.4}$$

The dot product of two vectors $\vec{a}$ and $\vec{b}$ is:

$$\vec{a} \cdot \vec{b} = \|a\| \, \|b\| \cos(\theta) \tag{3.5}$$

We can compute the angle θ between $\vec{a}$ and $\vec{b}$ from this equation. The cross product is only meaningful in three dimensions. The cross product differs from the dot product primarily in the

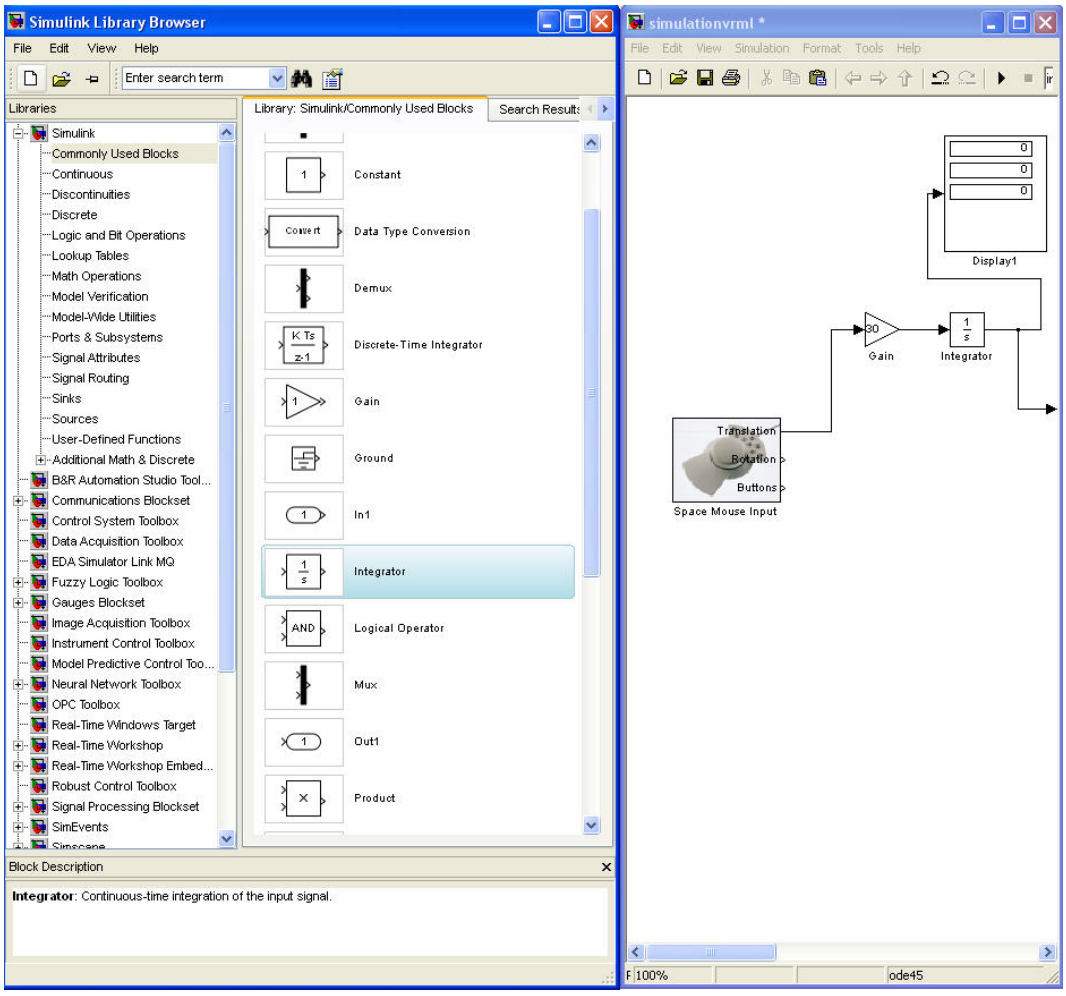


Figure 3.5.:Change the speed signal into distance signal during use Integrator block

way, that the result of the cross product of two vectors is a vector. The cross product, denoted $\vec{a}$, $\vec{b}$, is a vector perpendicular to both a and b and is defined as :

$$\vec{a} \times \vec{b} = \|a\| \, \|b\| \sin(\theta)\vec{n} \tag{3.6}$$

The geometric meaning of cross product is described by Fig. 3.10:

Now, we can use this mathematical basis to develop the code for motion using Matlab (see Fig. 3.11):

We gave the object a position in the VRML world. Then we used the 3D mouse to change the position of the camera, so we got two vectors and we needed a direction vector for the camera. We computed the vector between the object and the camera, then we needed to let the camera focus on the object. First of all, we known the original direction of camera in VRML world $[0, 0, 1]^T$ and because of the different of coordinate systems (see Fig. 2.1), the direction of camera out of VRML world was $[0, 0, -1]^T$. So the cross product of the direction vector of camera $[0, 0, -1]^T$ and the vector between camera and the object was computed, then the camera was rotated about this cross product to tracking on the object. The angle of the rotation about

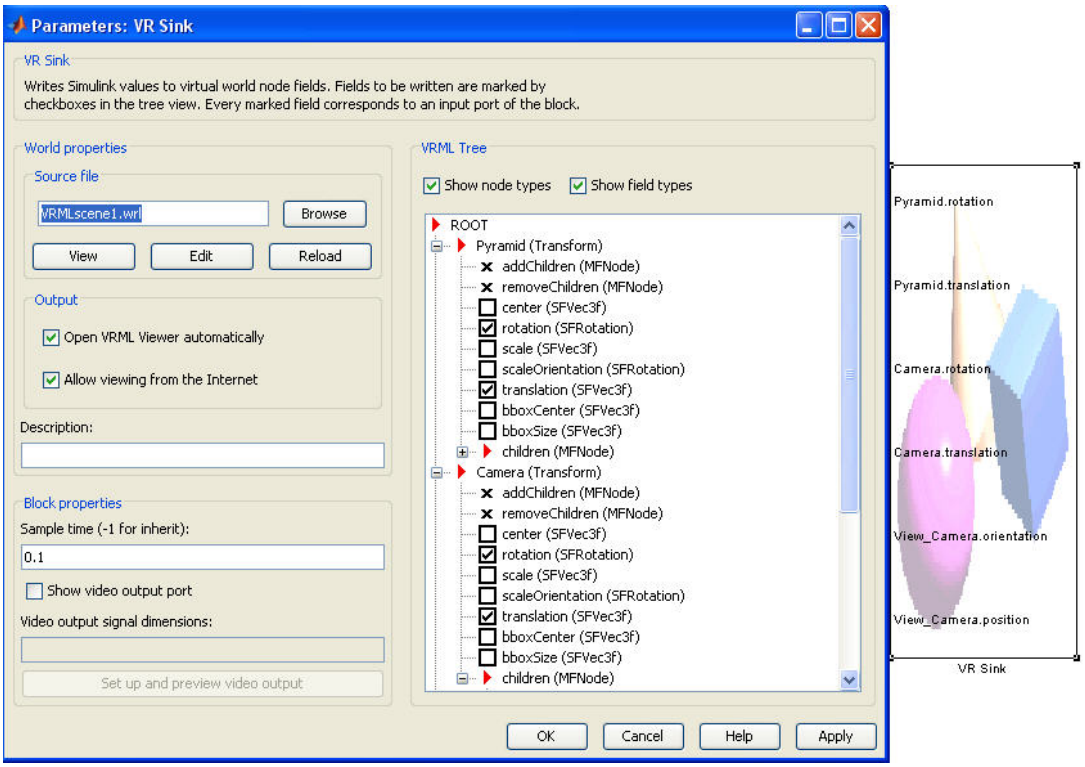


Figure 3.6.:Choosing the ports to write data to the virtual world

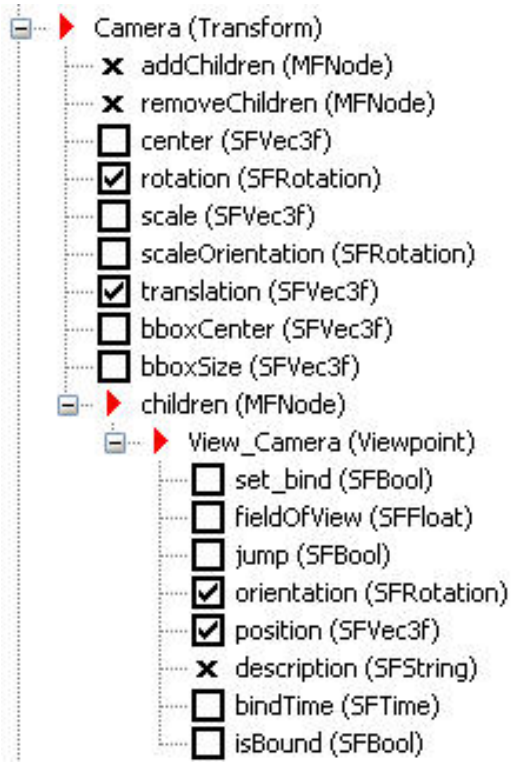


Figure 3.7.:Two types of the movement for the camera

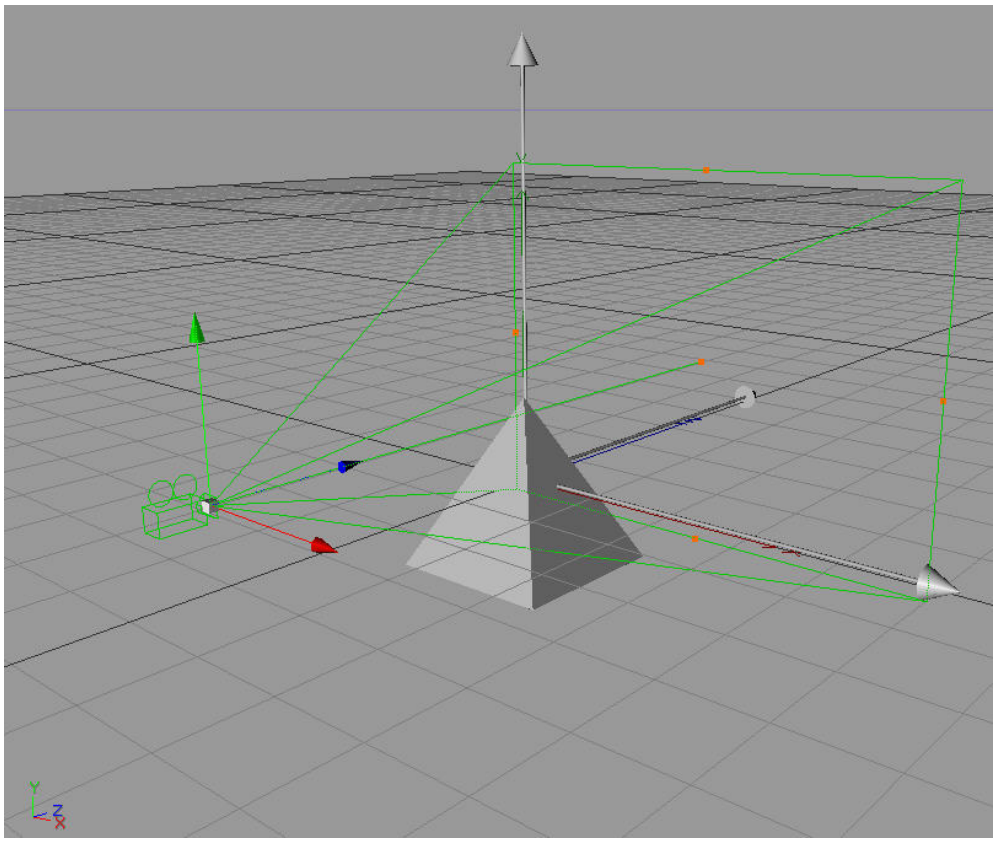


Figure 3.8.:Position of the camera in the virtual world

this axis was computed using the dot product (Equ. 3.5). In this code we used a "normalize" block, which takes an input vector of any size and outputs the unit vector parallel to it [22], to get the unit vector for the cross product. The final product realised the tracking, this means, we changed the position of the camera but the camera always focused on the object. Or we changed the position of the object, and the camera was always focused on the object.

3.4. Roll, Pitch and Yaw

In this section we want to let the camera rotate about axes of the own coordinate system and translate it in the own coordinate system. To solve this problem we need to known the mathematics of Roll, Pitch and Yaw. When an object rotates about the x-axis (see Fig. 3.12), we get following equations:

$$\begin{aligned} \mathrm{x}' &= \mathrm{x} \\ \mathrm{y}' &= \mathrm{y}\cos(\theta) - \mathrm{z}\sin(\theta) \\ \mathrm{z}' &= \mathrm{y}\sin(\theta) + \mathrm{z}\cos(\theta) \end{aligned} \tag{3.7}$$

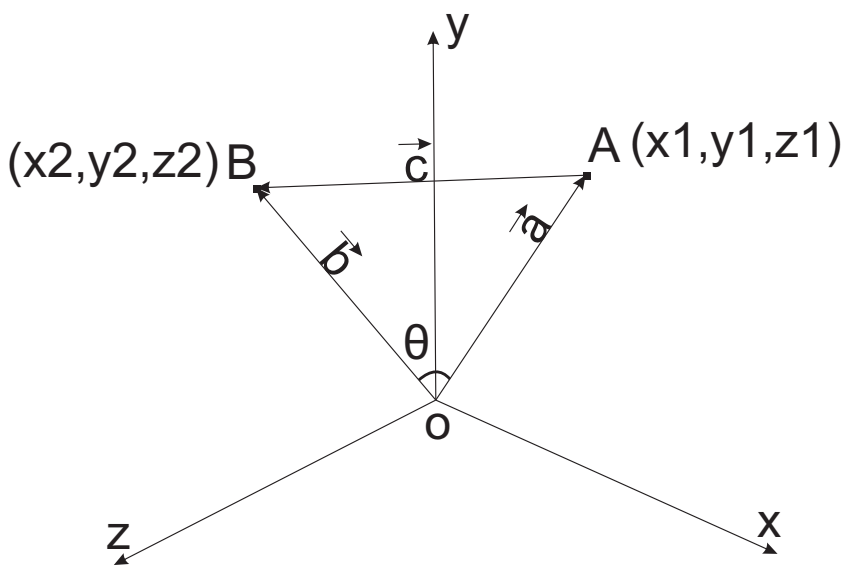


Figure 3.9.:Direction vector of camera and object in the global coordinate system

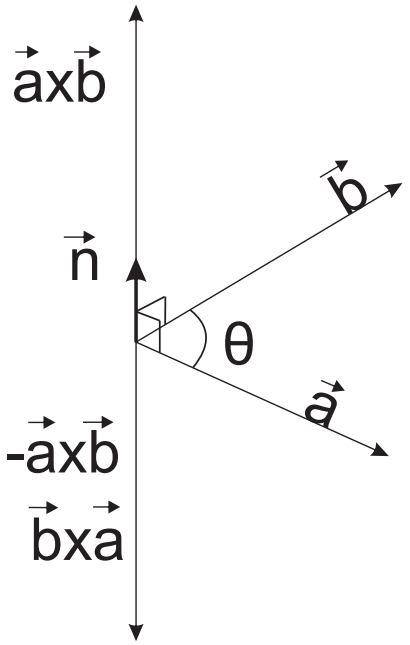


Figure 3.10.:Cross product of vectors

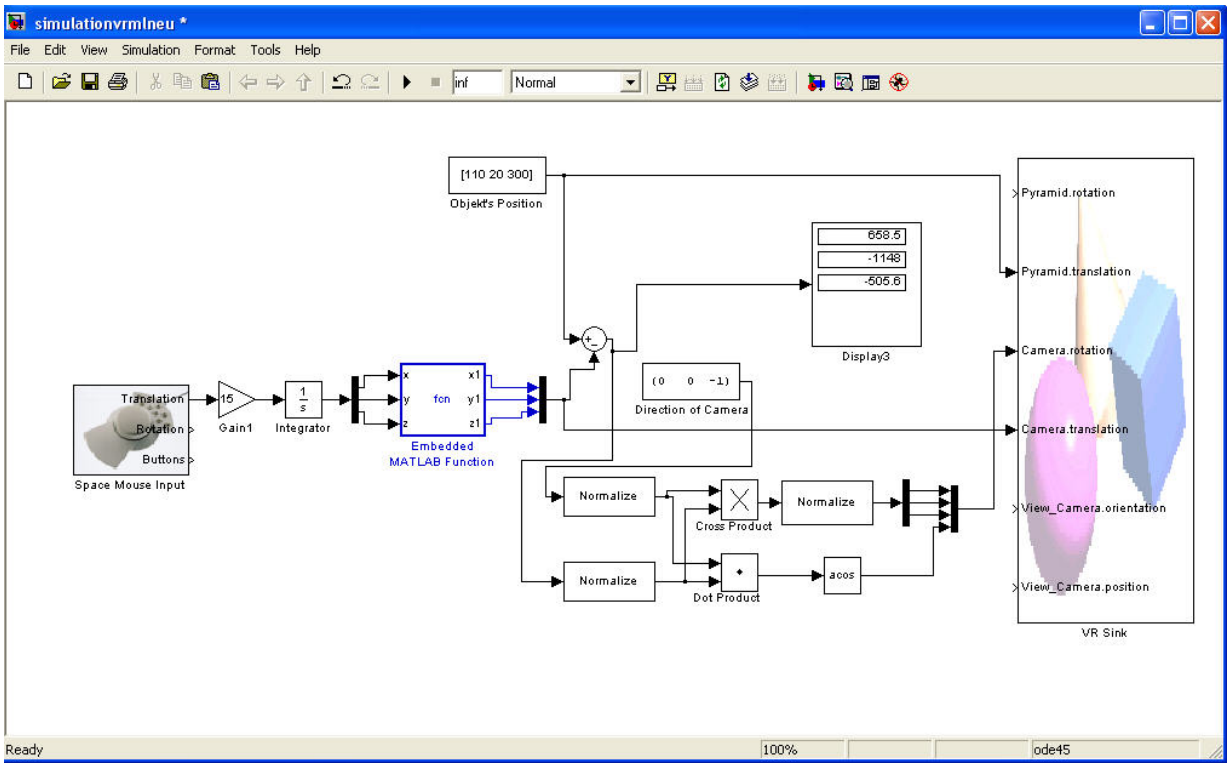


Figure 3.11.:Matlab Code for the camera tracking

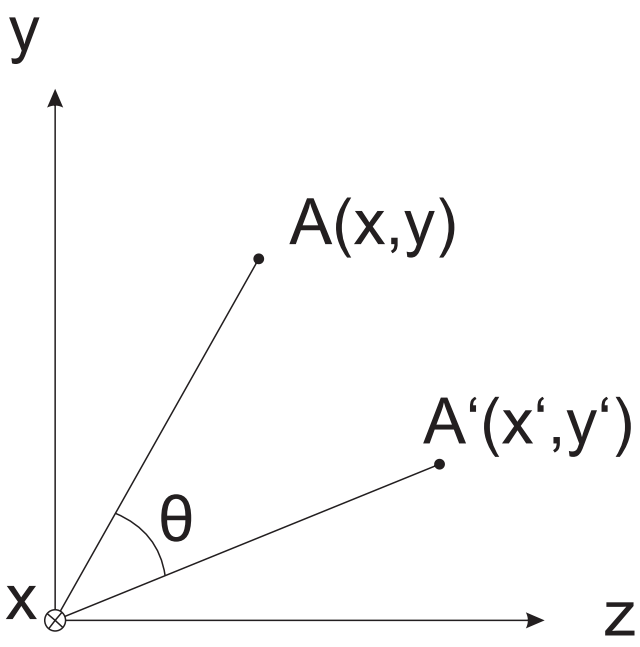


Figure 3.12.:Rotation about x-axis

The rotation matrix from equation 3.7 is:

$$Rx = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix} \tag{3.8}$$

As we know, the rotation about z-axis is yaw, the rotation about y-axis is pitch, the rotation about x-axis is roll. First we want to solve this problem with rotation matrices of Roll, Pitch and Yaw. We multiplied these three rotation matrixes $R_x \cdot R_y \cdot R_z$, which is the same as equation 3.9, to change the input signal to these three rotation. From equation 3.9 we found a problem, that the rotations must have the same sequence as $R_x \cdot R_y \cdot R_z$, this means the first rotation must about x-axis and the last rotation must be about z″-axis (see Fig 3.13), finally only the z″-axis was the correct axis of the new coordinate system, which after rotated about x-axis and y′-axes (see Fig. 3.13). This means the object can not rotate about x″-axis and y″-axis, if we want let the object rotate about them we need multiply an other R_x or R_y to equation 3.9, this means if we need a rotation about new axis we must multiply a new rotation matrix behind equation what we already had. This method makes an endless equation. If $M = R_x \cdot R_y \cdot R_z$ and $M' = R_z \cdot R_y \cdot R_x$. M and M′ are non-commutative.

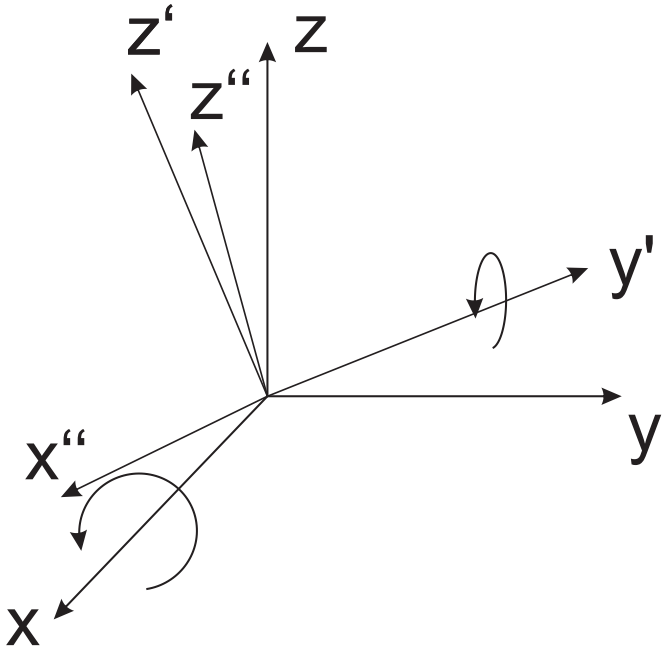


Figure 3.13.:Coordinate system rotating about axes

Now, the problem of the rotation is how we can get the new rotation matrix, and how we can get the rotation about the new axis. According to the mathematics of rotation with matrices

described previously, we need to multiply the rotation matrix and the rotation matrix for x, y and z axis:

$$M' = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\alpha') & \sin(\alpha') \\ 0 & -\sin(\alpha') & \cos(\alpha') \end{bmatrix} \begin{bmatrix} \cos(\beta') & 0 & -\sin(\beta') \\ 0 & 1 & 0 \\ \sin(\beta') & 0 & \cos(\beta') \end{bmatrix} \begin{bmatrix} \cos(\gamma') & \sin(\gamma') & 0 \\ -\sin(\gamma') & \cos(\gamma') & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.9}$$

M' is a rotated matrix, we multiply this rotated matrix with the new rotation matrix:

$$M'' = M' \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\alpha) & \sin(\alpha) \\ 0 & -\sin(\alpha) & \cos(\alpha) \end{bmatrix} \begin{bmatrix} \cos(\beta) & 0 & -\sin(\beta) \\ 0 & 1 & 0 \\ \sin(\beta) & 0 & \cos(\beta) \end{bmatrix} \begin{bmatrix} \cos(\gamma) & \sin(\gamma) & 0 \\ -\sin(\gamma) & \cos(\gamma) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.10}$$

Now, we get the rotation about new x, y and z axis, if we need always rotation about the new axis, we need a cycle of this multiplication. The problem to do this for our project can not be solved, because we have only one input for three rotation angles, after each rotation we need to set the value of the rotation angel to zero. We can resolve this problem with speed of the input signal, but the problem is how can we get the multiplication like function M''. So we need to solve this problem with another method.

We tried to solve this problem with a vector, as we know the rotation axis is a vector in the own coordinate system of the camera, the new rotation axes is a vector in the primary own coordinate system. If we can compute the new direction vector of the rotation axis with angular velocity of the rotation and the primary vector of the x, y and z axis in primary coordinate system, we can get the new coordinate system.

First, we studied how to change the position of axis after rotation (see Fig. 3.14-3.16), if the rotation angle is small enough, we can get the direction of de_x, de_y and de_z have the same direction as e_x, e_y and e_z, then we used the differential equation for roll, pitch and yaw.

The direction of axis is changed by roll:

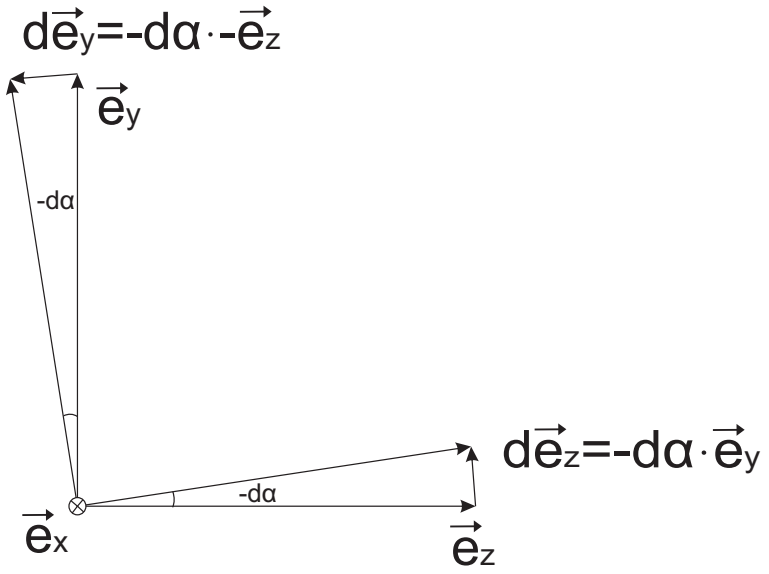


Figure 3.14.:Direction of axis is changed during rotation about x-axis

The direction of axis is changed by pitch:

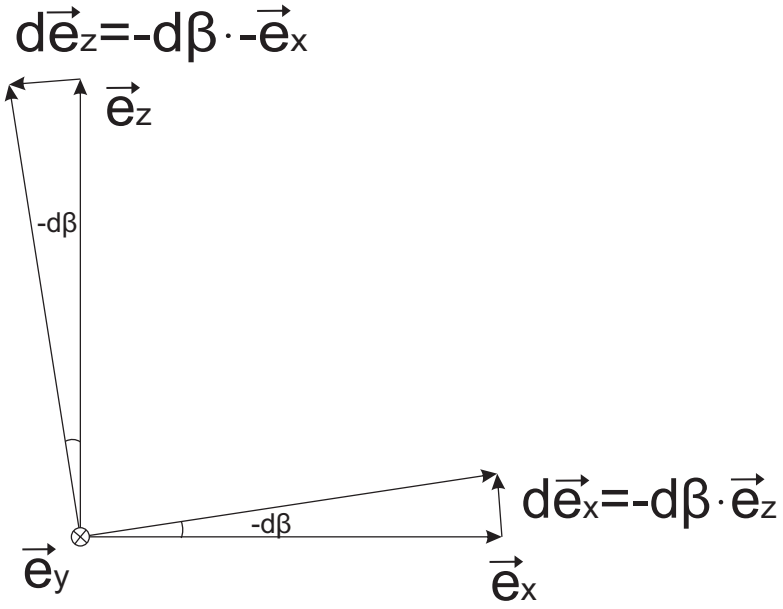


Figure 3.15.:Direction of axis is changed during rotation about y-axis

The direction of axis is changed by yaw:

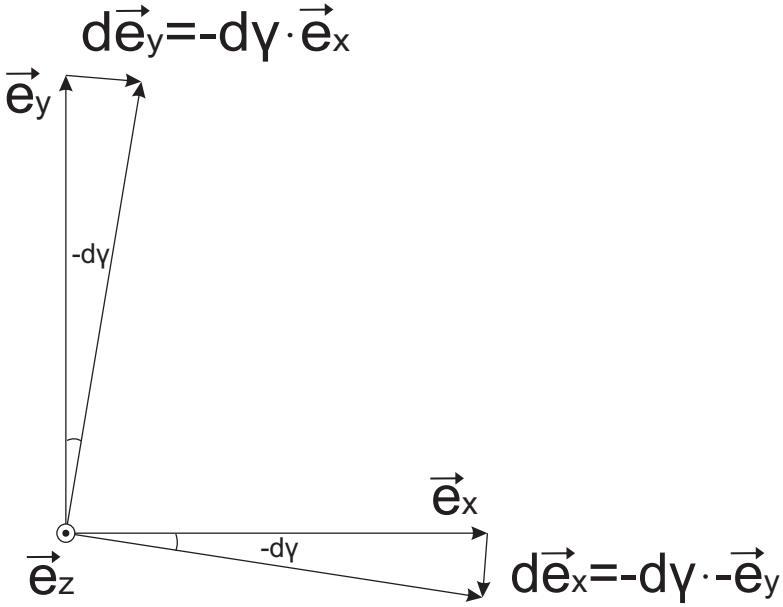


Figure 3.16.:Direction of axis is changed during rotation about z-axis

Then, we computed the differential equation for three axes from equations 3.11, 3.12 and 3.13:

$$d\vec{e_x} = \dot{\vec{e_x}} = \dot{\gamma} \cdot \vec{e_y} - \dot{\beta} \cdot \vec{e_z} \tag{3.11}$$

$$d\vec{e_y} = \dot{\vec{e_y}} = \dot{\alpha} \cdot \vec{e_z} - \dot{\gamma} \cdot \vec{e_x} \tag{3.12}$$

$$d\vec{e_z} = \dot{\vec{e_z}} = \dot{\beta} \cdot \vec{e_x} - \dot{\alpha} \cdot \vec{e_y} \tag{3.13}$$

Because of the type of our signal, differential equations describe the velocities, which are results of the rotations about axis of coordinate system, which changes the direction vector of axis. We built this function with the Matlab/Simulink-Subsystem in Fig. 3.17.

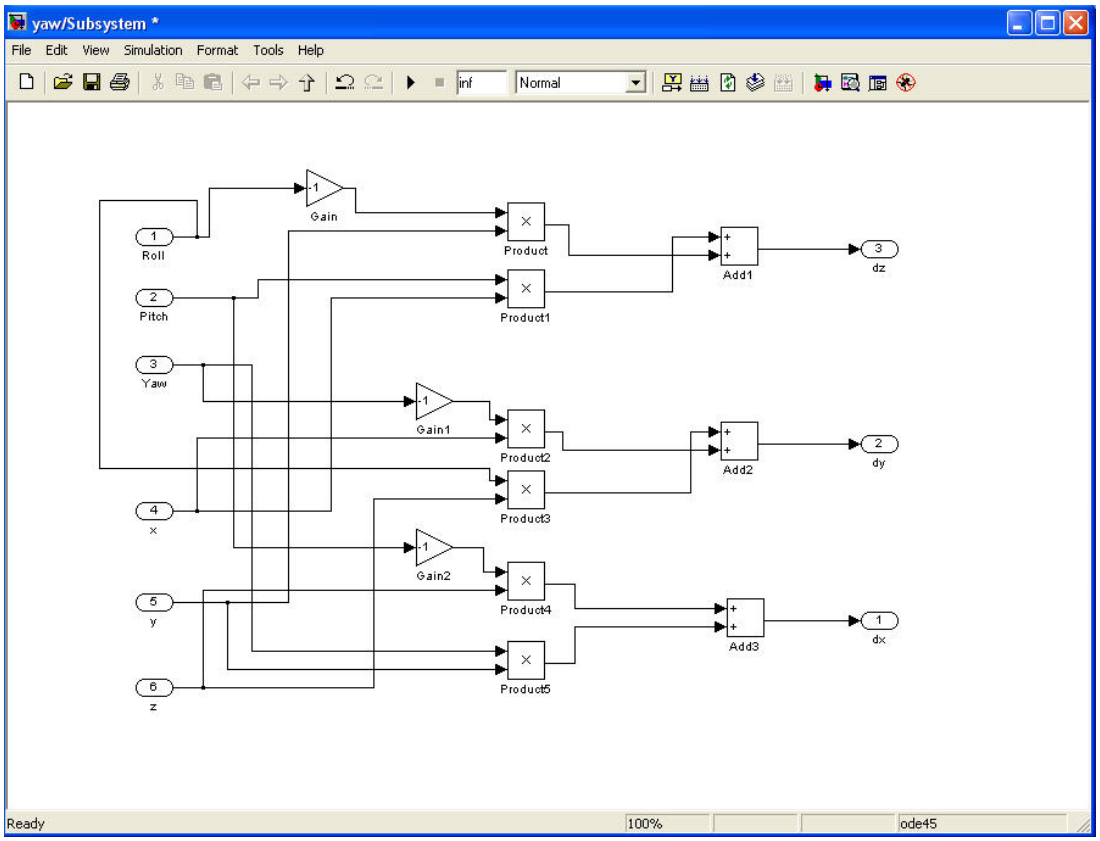


Figure 3.17.:Matlab code for the differential equation of rotation

Then we built Matlab code for roll, pitch and yaw, the input signals for the functions in Fig. 3.17 are velocities.

The following vectors are initial positions of direction vectors for the initial coordinate system. This initial condition is the roll axis in the primary coordinate system:

$$\vec{e_x} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \tag{3.14}$$

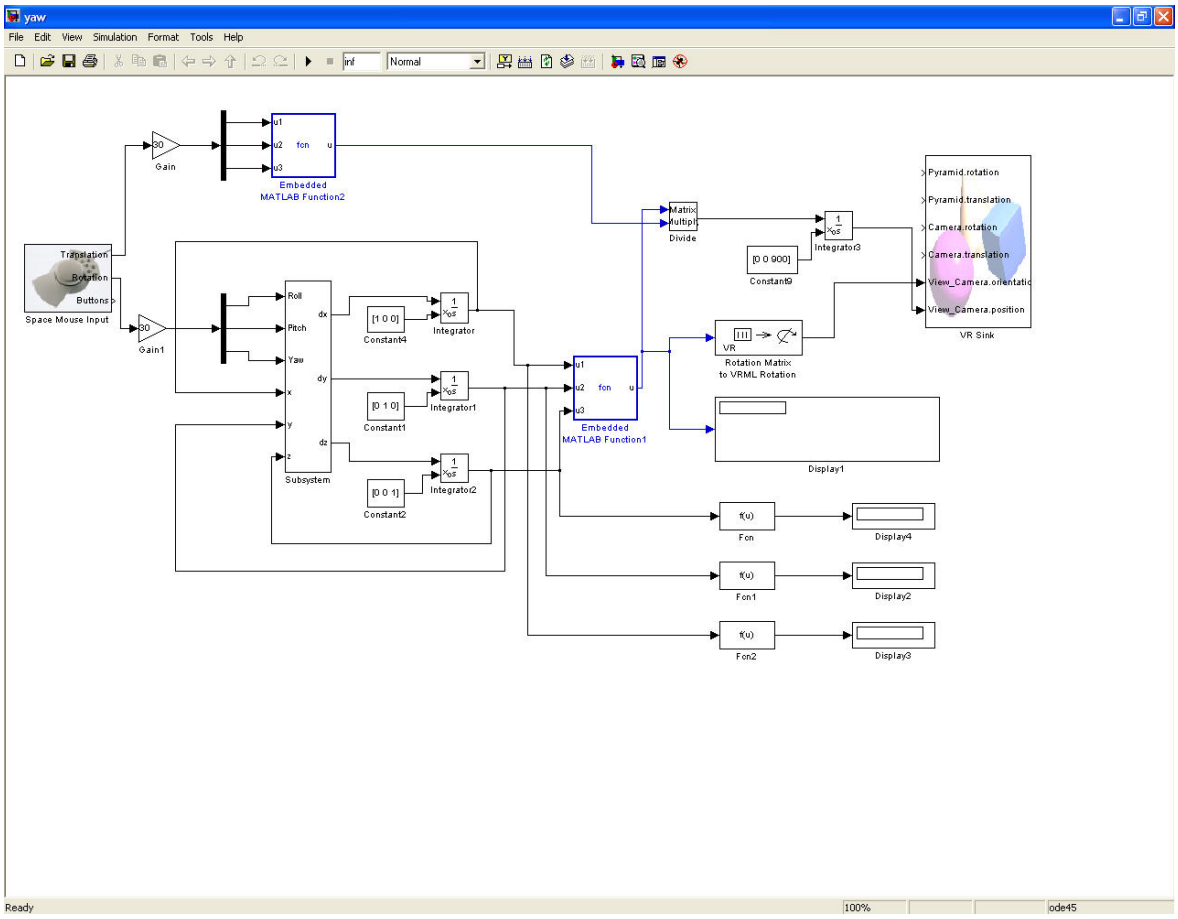


Figure 3.18.:Matlab code for Roll, Pitch and Yaw

This initial condition is for pitch axis in the initial position of coordinate system:

$$\vec{e_y} = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \tag{3.15}$$

This initial condition is for yaw in the primary coordinate system:

$$\vec{e_z} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \tag{3.16}$$

We used the Integrator block, which integrate the inputs with the initial condition of direct vector [22], to compute the new direction vector of the new axis. The position of initial coordinate system was changed from the initial condition after rotations, then we gave the changed value of the axis back to the subsystem for the differential equation of rotation. After this, the direction vectors of the coordinate system were the new coordinate system. On this basis, we got the new axis in initial coordinate system during changing velocity of the x, y and z values during integration. After integration of this equation, we got the new unit vector of the three axes, this means the three axes of the coordinate system was changed over time.

It should be mentioned, that numerical solution of differential equations leads to accumulation of computing errors. Consequently, this method is not suited for infinite operation without additional measures. To check the motion of the axes, we needed a function to compute the length of the unit vectors e_x, e_y and e_z:

$$|\vec{e_x}| = |\vec{e_y}| = |\vec{e_z}| = \sqrt{x^2 + y^2 + z^2} = 1 \tag{3.17}$$

We wrote a Matlab code for this function using Fcn block (see Fig. 3.19), which applies the specified mathematical expression to its input [22]:

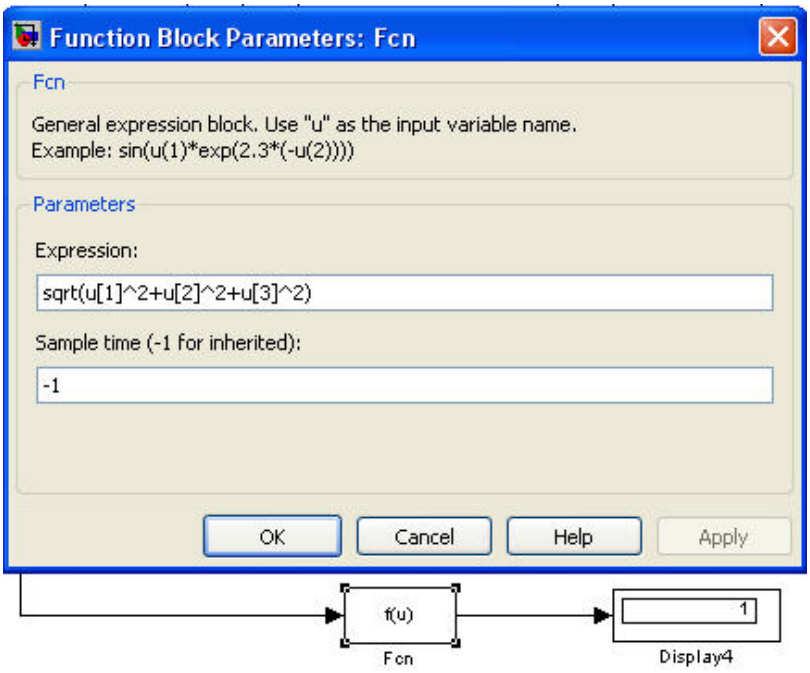


Figure 3.19.:Compute the length of the unit vector

As we described in chapter 2, we have only one port for the control of the camera rotation in VRML format, the problem is how can we write all direction vectors into VRML world. We found a method, which includes all three unit vector of the axis, to solve this problem. A matrix can be written as [12]:

$$\begin{bmatrix} a_{0.0} & a_{0.1} & a_{0.2} \\ a_{1.0} & a_{1.1} & a_{1.2} \\ a_{2.0} & a_{2.1} & a_{2.2} \end{bmatrix} \tag{3.18}$$

We can represent its rows as three vectors:

$$\begin{aligned} a_0^T &= \begin{pmatrix} a_{0.0} & a_{0.1} & a_{0.2} \end{pmatrix} \\ a_1^T &= \begin{pmatrix} a_{1.0} & a_{1.1} & a_{1.2} \end{pmatrix} \\ a_2^T &= \begin{pmatrix} a_{2.0} & a_{2.1} & a_{2.2} \end{pmatrix} \end{aligned} \tag{3.19}$$

Then the matrix can be written like this:

$$\begin{bmatrix} a_0^T \\ a_1^T \\ a_2^T \end{bmatrix} \tag{3.20}$$

Similarly, we can represent a matrix with its columns as three vectors [12]:

$$b_0 = \begin{pmatrix} b_{0.0} \\ b_{1.0} \\ b_{2.0} \end{pmatrix}$$

$$b_1 = \begin{pmatrix} b_{0.1} \\ b_{1.1} \\ b_{2.1} \end{pmatrix} \tag{3.21}$$

$$b_2 = \begin{pmatrix} b_{0.2} \\ b_{1.2} \\ b_{2.2} \end{pmatrix}$$

The matrix can be written as:

$$\begin{bmatrix} b_0 & b_1 & b_2 \end{bmatrix} \tag{3.22}$$

The initial coordinate system and the new coordinate system are described (see Fig. 3.20):

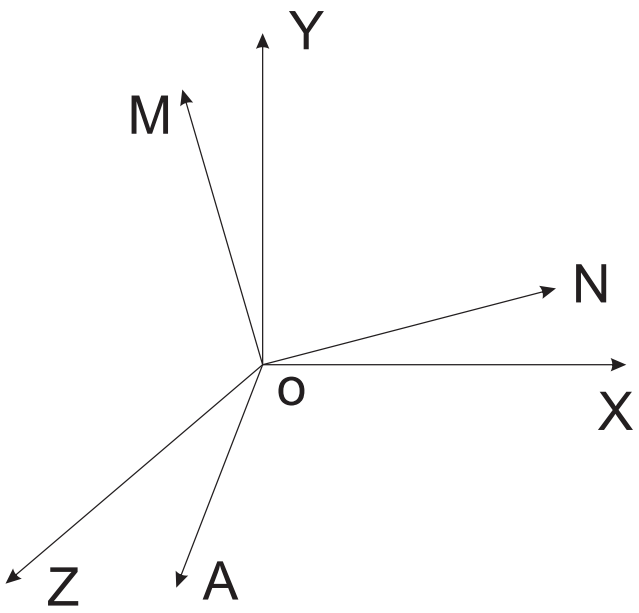


Figure 3.20.:Primary coordinate system and rotated coordinate system

The unit vectors for the rotated coordinate system are:

$$\begin{aligned}\overrightarrow{N} &= \begin{pmatrix} n_x & n_y & n_z \end{pmatrix} \\ \overrightarrow{M} &= \begin{pmatrix} m_x & m_y & m_z \end{pmatrix} \\ \overrightarrow{A} &= \begin{pmatrix} a_x & a_y & a_z \end{pmatrix}\end{aligned} \tag{3.23}$$

The matrix can be written as:

$$\begin{bmatrix} n_x & m_x & a_x \\ n_y & m_y & a_y \\ n_z & m_z & a_z \end{bmatrix} \tag{3.24}$$

This matrix describes rotation about X-axis, Y-axis, Z-axis or a combination of the three, we set the unit vector of the axis in this matrix as three columns like:

$$R = \begin{bmatrix} u1_1 & u2_1 & u3_1 \\ u1_2 & u2_2 & u3_2 \\ u1_3 & u2_3 & u3_3 \end{bmatrix} \tag{3.25}$$

where, $u1$, $u2$ and $u3$ are inputs to Matlab Function 1 (see Fig. 3.21).

To test this method, we built the Matlab code (see Fig. 3.21): Then we fed the rotation matrix

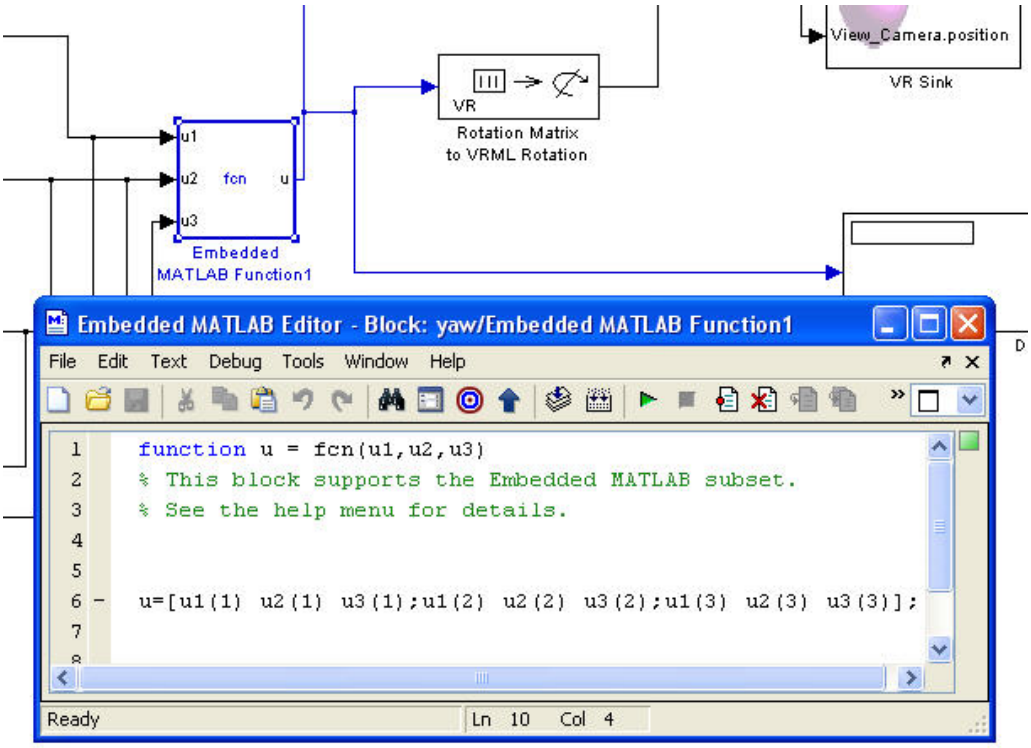


Figure 3.21.:Changing matrix to VRML rotation

to the VRML Rotation block, which takes a rotation matrix as input and outputs the axis/angle rotation representation used for defining rotations in VRML, to test our method [22]. Finally, we got exactly the correct result, this means we got the correct rotation, which was about the new coordinate system.

Then we translated the camera in VRML world using the 3D mouse, which gave the x, y and z values. It is no problem before the own coordinate system of the camera changed, this means orientation of the camera was not changed. Then we changed the orientation of the camera and found that, we could only change the x, y and z values along axes of initial coordinate system. This means we could not change the position of the camera at the correct direction. To resolve this problem we needed to know the mathematics of translation and rotation. We used mathematics what we described in chapter 2 to resolve this problem. We already knew the new position of the coordinate system of camera. When we multiplied the rotation matrix and initial x, y and z values like equation 3.26, we solved this problem. The change of the x, y and z values are following the new x, y, z axis, but there was another problem. Since the type of signal is speed, we needed an Integrator, when we used it before the multiplication of x, y and z values with rotated matrix. When we changed the rotation matrix, the value of the translation was also changing with it. Because of this problem, the rotation of the camera was about the wrong axis, not the new axis. When we used it after the multiplication, the change of the velocity of the x, y and z values is correct, so we solved the problem (see Fig. 3.22).

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} u1_1 & u2_1 & u3_1 \\ u1_2 & u2_2 & u3_2 \\ u1_3 & u2_3 & u3_3 \end{bmatrix} \cdot \begin{bmatrix} t1 \\ t2 \\ t3 \end{bmatrix} \tag{3.26}$$

Matlab code for the translation at new coordinate system. Finally, we could control the motion

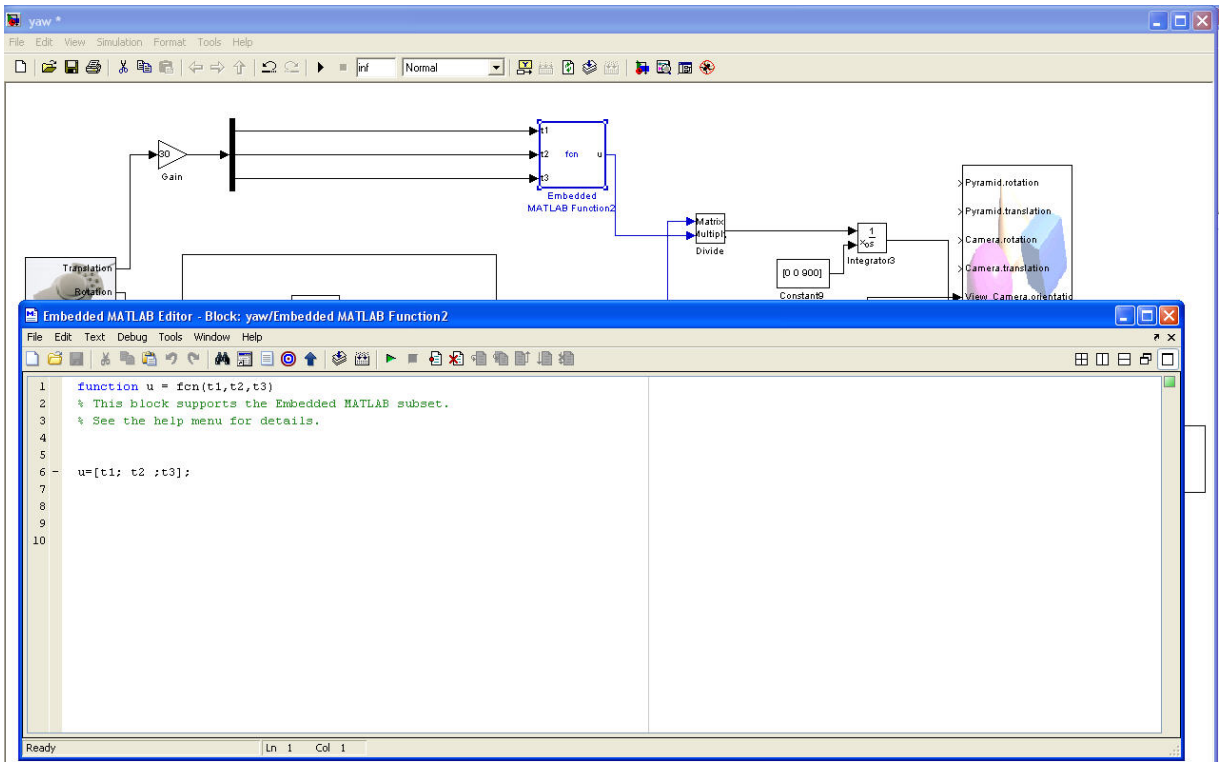


Figure 3.22.:Matlab code for the translation

of camera in VRML world well with the developed Matlab code. After studying mathematics of motion in VRML world, we could change the position of the camera not only in global coordinate system, also in the own coordinate system.

4. Camera Motion in a Virtual World Using a Virtual Robot

In the previous chapter we developed the Matlab code for the motion of the camera in VRML world. Sometimes we want to control a camera in real world like in VRML world. For this purpose we can use a robot, which can achieve the motion of camera. First of all, we can study the motion of camera with a virtual robot. In this chapter we will study the transforms of camera with a 6DOF robot with six axes, which means it can reach any point of a sphere, when we want to get a point of the sphere, we need to understand the mathematics of transformation and the kinematics of the robot.

4.1. Denavit-Hartenberg Parameters

If we want to describe the motion of a serial robot with swivel joint, we can use the Denavit-Hartenberg Parameters. A commonly used convention for selecting frames of reference in robotics applications is the Denavit and Hartenberg (DH) convention, which was introduced by Jaques Denavit and Richard S. Hartenberg in the year 1955 [32]. To find a minimal representation, the common normal between two lines is the main geometric concept used by Denavit and Hartenberg. The frame are laid out as follows [32]:

- Z_n-axis is in the direction of the joint axis.
- X_n-axis is as :

$$X_n = Z_{n-1} \times Z_n \tag{4.1}$$

If there is no unique common normal (parallel z axes), then the offset d is a free parameter.
- The y-axis follows from the x- and z-axis by choosing it to be a right-handed coordinate system.

Then the transformation is described by the following three Denavit-Hartenberg parameters [11]:

- α_n: angle from g_i-axis to g_{i+1}-axis.
- d_n: offset along g_i to the common normal between g_i and g_{i+1}.
- a_n: length of the common normal from g_i-axis to g_{i+1}-axis.

We looked for the Denavit-Hartenberg parameters of our 6-axes serial robot (see Fig. 4.1).

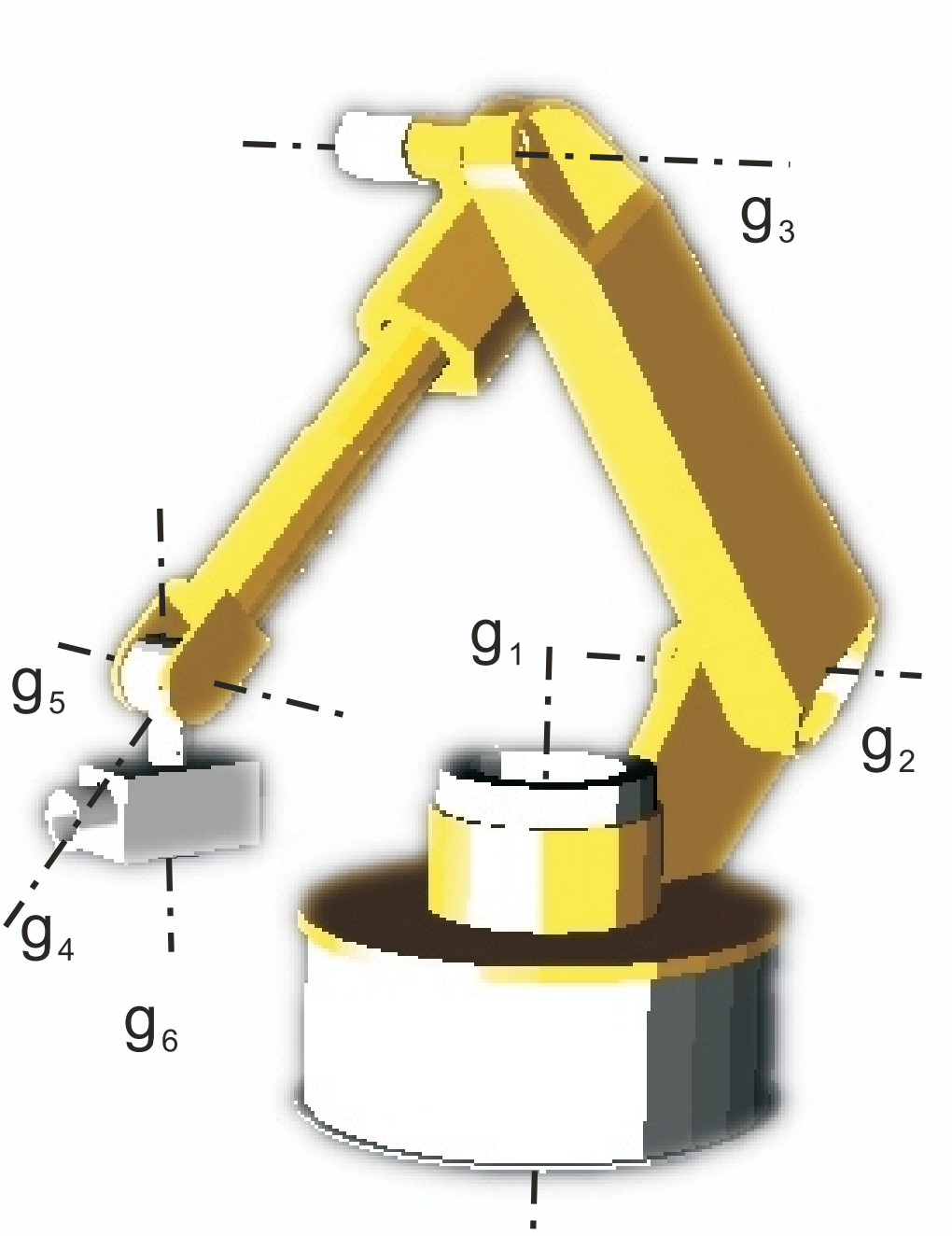


Figure 4.1.:Rotation axes of our serial robot [1]

[1]This virtual robot is from the demo of Matlab 2009b/SimMechanics [23].

First of all, we built a 6-axes serial robot with Matlab (see Fig. 4.2).

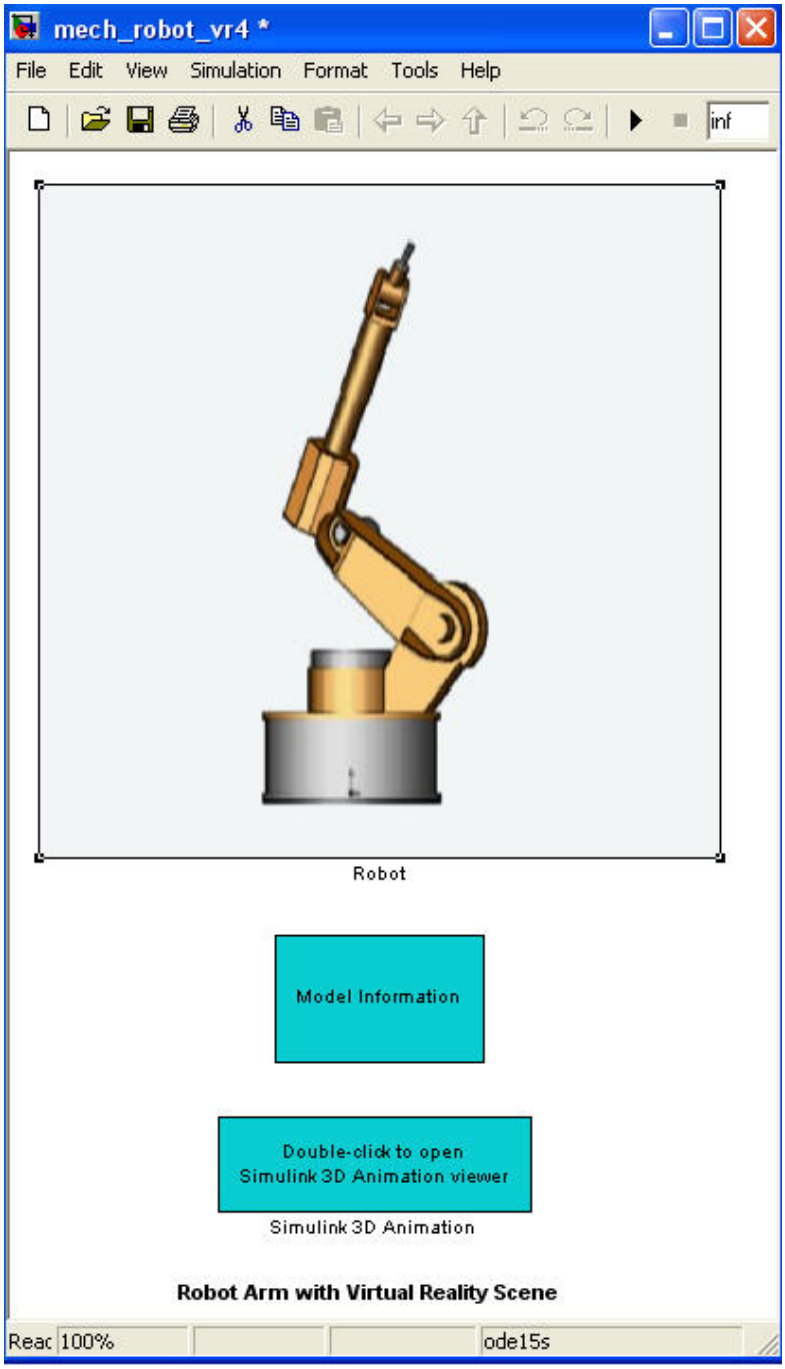


Figure 4.2.:6-axes serial robot [1]

[1] This robot arm with virtual reality scene is from the demo of Matlab 2009b/SimMechanics [23].

This serial robot has six bodies (see Fig. 4.3), we must fix all of them with coordinates of center of joints in VRML world (see Fig. 4.4):

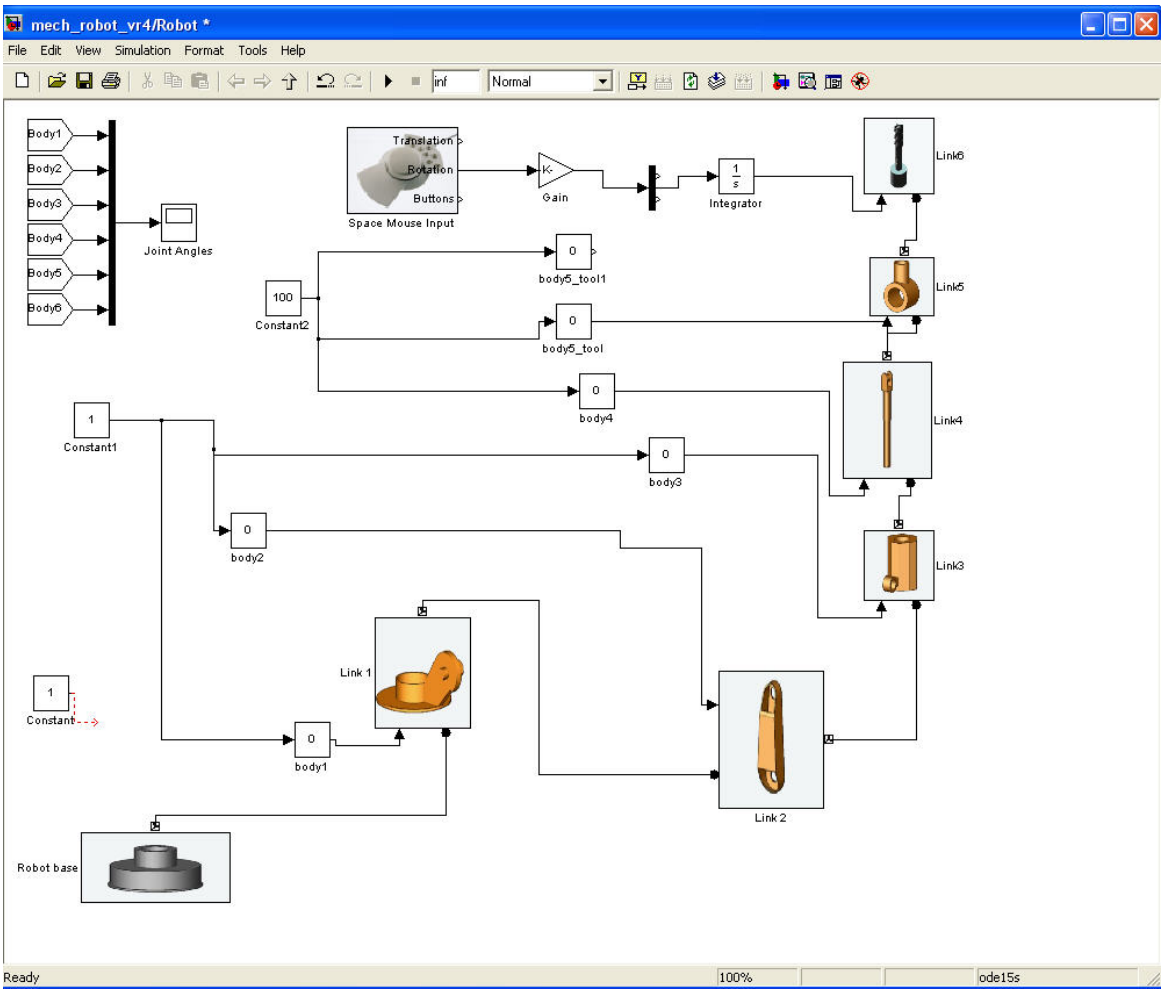


Figure 4.3.:6 bodies of the serial robot [1]

The center coordinates of the first joint are the origin point of the base coordinate system ($e_{0,1}$,$e_{0,2}$,$e_{0,3}$) (see Fig. 4.6) and are based on the global coordinate system (coordinate system of VRML world):

$$\begin{bmatrix} z_1 \\ y_1 \\ x_1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 3000 \end{bmatrix} \tag{4.2}$$

Because we only study the motion of robot, we let the coordinates of the origin of the base coordinate system be:

$$\begin{bmatrix} z_1 \\ y_1 \\ x_1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{4.3}$$

[1] This robot arm with virtual reality scene is from the demo of Matlab 2009b/SimMechanics [23].

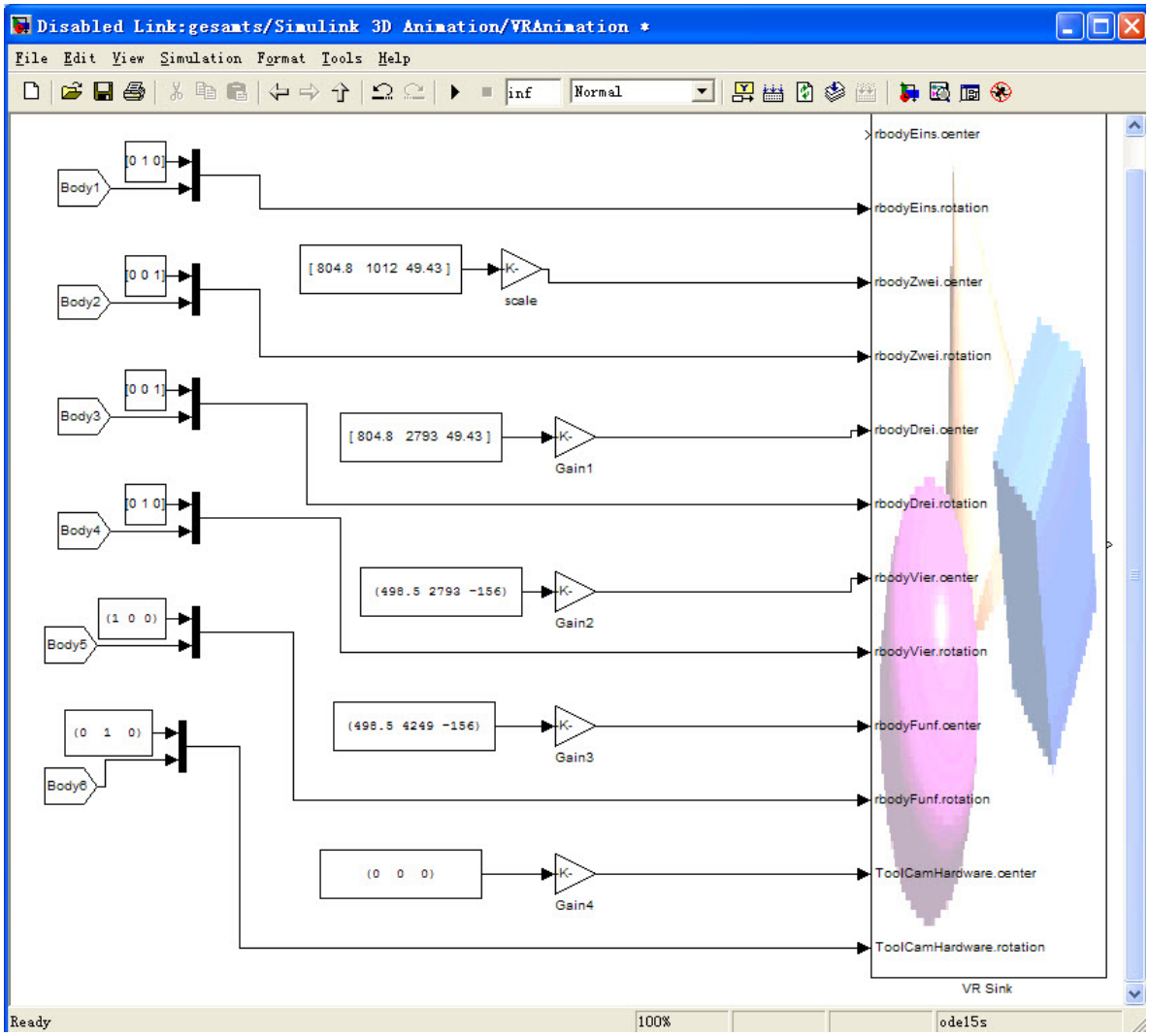


Figure 4.4.:Coordinates of the centers of joints

The coordinates of center of the second joint are:

$$\begin{bmatrix} z_2 \\ y_2 \\ x_2 \end{bmatrix} = \begin{bmatrix} 804.8 \\ 1012 \\ 49.43 \end{bmatrix} \tag{4.4}$$

The coordinates of center of the third joint are:

$$\begin{bmatrix} z_3 \\ y_3 \\ x_3 \end{bmatrix} = \begin{bmatrix} 804.8 \\ 2793 \\ 49.43 \end{bmatrix} \tag{4.5}$$

The coordinates of center of the fourth joint are:

$$\begin{bmatrix} z_4 \\ y_4 \\ x_4 \end{bmatrix} = \begin{bmatrix} 498.5 \\ 2793 \\ -156 \end{bmatrix} \tag{4.6}$$

The coordinates of center of the fifth joint are:

$$\begin{bmatrix} z_5 \\ y_5 \\ x_5 \end{bmatrix} = \begin{bmatrix} 498.5 \\ 4249 \\ -156 \end{bmatrix} \tag{4.7}$$

Then we computed all Denavit-Hartenberg parameters of our robot from equations 4.4-4.7:

α_1=$\frac{\pi}{2}$, a_1=804.8, d_1=1012;
α_2=0, a_2=1781, d_2=0;
α_3=$\frac{\pi}{2}$, a_3=306.3, d_3=205.43;
α_4=$\frac{\pi}{2}$, a_4=0, d_4=1456;
α_5=$\frac{\pi}{2}$, a_5=0, d_5=0;
Because of $e_{2,3}$ and $e_{1,3}$ are not collinear, this means the length of common normal between $e_{2,3}$ and $e_{1,3}$ is $a_1' = 49.43$.

If we want to control the motion of the robot, we must study kinematics of the robot and find a solution for our project.

4.2. Kinematics

Kinematics is the formal description of motion. One of the goals of rudimentary mechanics is to identify forces on a point object and then apply kinematics to determine the motion of the object. Ideally the position of the object at all times can be determined. For an extended object, (rigid body or other), along with linear kinematics, rotational motion can be applied to achieve the same objective: Identify the forces, develop the equations of motion, find the position of center of mass and the orientation of the object at all times [32]. Robot Kinematics mainly has two types, forward kinematics (direct kinematics) and inverse kinematics.

4.2.1. Forward Kinematics

Forward kinematics is computation of the position and orientation of robot's end effector as a function of its joint angles [32]. First, we need to know that the 6-axes robot has six local coordinate systems and a base coordinate system, we can give every transform of each coordinate system, then we get the position of the end coordinate system in the base coordinate system, we call this forward kinematics, or, we first know the position of the latest coordinate system in the global coordinate system, then we can solve every transform of each coordinate system, we call this inverse kinematics.

If we use forward kinematics, we must know Denavit-Hartenberg (DH) convention, which is already described in section 1. In this convention, each homogeneous transformation is represented as a product of four basic transformations. The common normal between two lines was the main geometric concept that allowed Denavit and Hartenberg to find a minimal representation. The precondition of Denavit-Hartenberg convention are [32]:
1. The coordinate system is fixed on rotation axis.
2. The z-axis is in the direction of the joint axis.
3. The x-axis is parallel to the common normal: $\vec{x}_n$=$\vec{z}_{n-1} \times \vec{z}_n$.
If there is no unique common normal (parallel z axes), then d is a free parameter.
4. The y-axis follows from the x- and z-axis by choosing it to be a right-handed coordinate system.

In DH convention a transformation is represented as concatenation of three transformations [11]:

$$\begin{bmatrix} x_i \\ y_i \\ z_i \end{bmatrix} = \underbrace{\begin{bmatrix} d_i \\ 0 \\ a_i \end{bmatrix}}_{T(x_i,y_i,z_i)} + \underbrace{\begin{bmatrix} \cos(\alpha_i) & -\sin(\alpha_i) & 0 \\ \sin(\alpha_i) & \cos(\alpha_i) & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{R(z_i,\alpha_i)} \underbrace{\begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(u_{i+1}) & -\sin(u_{i+1}) \\ 0 & \sin(u_{i+1}) & \cos(u_{i+1}) \end{bmatrix}}_{R(x_{i+1},u_{i+1})} \begin{bmatrix} x_{i+1} \\ y_{i+1} \\ z_{i+1} \end{bmatrix} \tag{4.8}$$

In DH convention a transformation $T(x_i, y_i, z_i)$, $R(z_i, \alpha_i)$ and $R(x_{i+1}, u_{i+1})$ can be also represented as two homogeneous matrices [11]:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ \mathrm{d_i} & \cos(\alpha_i) & -\sin(\alpha_i) & 0 \\ 0 & \sin(\alpha_i) & \cos(\alpha_i) & 0 \\ a_i & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \cos(u_{i+1}) & -\sin(u_{i+1}) \\ 0 & 0 & \sin(u_{i+1}) & \cos(u_{i+1}) \end{bmatrix} \tag{4.9}$$

Where the four quantities α_i, $\mathrm{a_i}$, $\mathrm{d_i}$, u_{i+1} are parameters associated with link i and joint i. The four parameters α_i, $\mathrm{a_i}$, $\mathrm{d_i}$ and u_{i+1} in equation 4.8 are generally given the names link twist, link offset, link length, and joint angle, respectively. These names derive from specific aspects of the geometric relationship between two coordinate frames [29]. On this basis, we will look at forward kinematics of 6-axes serial robot, the coordinate of a point which based on end coordinate system must be given, then we give six rotation angles $u_1, ..., u_6$ of each axis, we can

get the coordinate of same point which based on the global coordinate system. The equation is [11]:

$$\begin{bmatrix} 1 \\ \mathrm{x}_0 \\ y_0 \\ z_0 \end{bmatrix} = B(u_1, ..., u_6) \cdot \begin{bmatrix} 1 \\ \mathrm{x}_6 \\ y_6 \\ z_6 \end{bmatrix} \tag{4.10}$$

Where $[x_0\ y_0\ z_0]^T$ is the coordinate of a point based on the global coordinate system. $[x_6\ y_6\ z_6]^T$ is the coordinate of same point based on the end effector coordinate system, $B(u_1...u_6)$ is a matrix, which depends on six rotation angles $u_1, ..., u_6$ and the Denavit-Hartenberg Parameters a_i, d_i, α_i [11]:

$$B(u_1, ..., u_6) = R_x(u_1) \cdot C_1 \cdot ... \cdot C_5 \cdot R_x(u_6) \tag{4.11}$$

Where:

$$C_i = \begin{bmatrix} 1 & 0 & 0 & 0 \\ \mathrm{d_i} & c_{\alpha_i} & -s_{\alpha_i} & 0 \\ 0 & s_{\alpha_i} & c_{\alpha_i} & 0 \\ a_i & 0 & 0 & 1 \end{bmatrix}, R_x(u_i) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & c_{\alpha_i} & -s_{\alpha_i} \\ 0 & 0 & s_{\alpha_i} & c_{\alpha_i} \end{bmatrix} \tag{4.12}$$

4.2.2. Inverse Kinematics

In this section, we will study inverse kinematics. Inverse kinematics is the process of determining the parameters of a jointed flexible object (a kinematic chain) in order to achieve a desired pose [32]. With the length of each link and position and orientation of the end effector of the robot given, we must compute the angles of each joint which are needed to obtain that position.

With this precondition, we must know the position of the end joint O_6 and the three vectors $e_{6,1}$, $e_{6,2}$ and $e_{6,3}$ (see Fig. 4.6), which are based on global coordinate system.

The position of the end joint based on the global coordinate system is [12]:

$$O_6 = \begin{bmatrix} b_{10} \\ b_{20} \\ b_{30} \end{bmatrix} \tag{4.13}$$

The three direction vectors based on the global coordinate system are [12]:

$$\begin{bmatrix} b_{11} \\ b_{21} \\ b_{31} \end{bmatrix}, \begin{bmatrix} b_{12} \\ b_{22} \\ b_{32} \end{bmatrix}, \begin{bmatrix} b_{13} \\ b_{23} \\ b_{33} \end{bmatrix} \tag{4.14}$$

Then we must compute six rotation angles using following equation [12]:

$$B(u1, u2...u6) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ b_{10} & b_{11} & b_{12} & b_{31} \\ b_{20} & b_{21} & b_{22} & b_{32} \\ b_{30} & b_{31} & b_{23} & b_{33} \end{bmatrix} \tag{4.15}$$

4.3. Camera Motion with a 6-Axes Serial Robot

In our project, we mounted a camera on the end effector of our virtual six axes serial robot, this means the camera is fixed on the end coordinate system. Then we wanted to control the motion of camera with a 3D mouse described in chapter 3 and watch in an own window, which displays the video image of the camera (see Fig. 4.5). First of all, if we want to control the position of

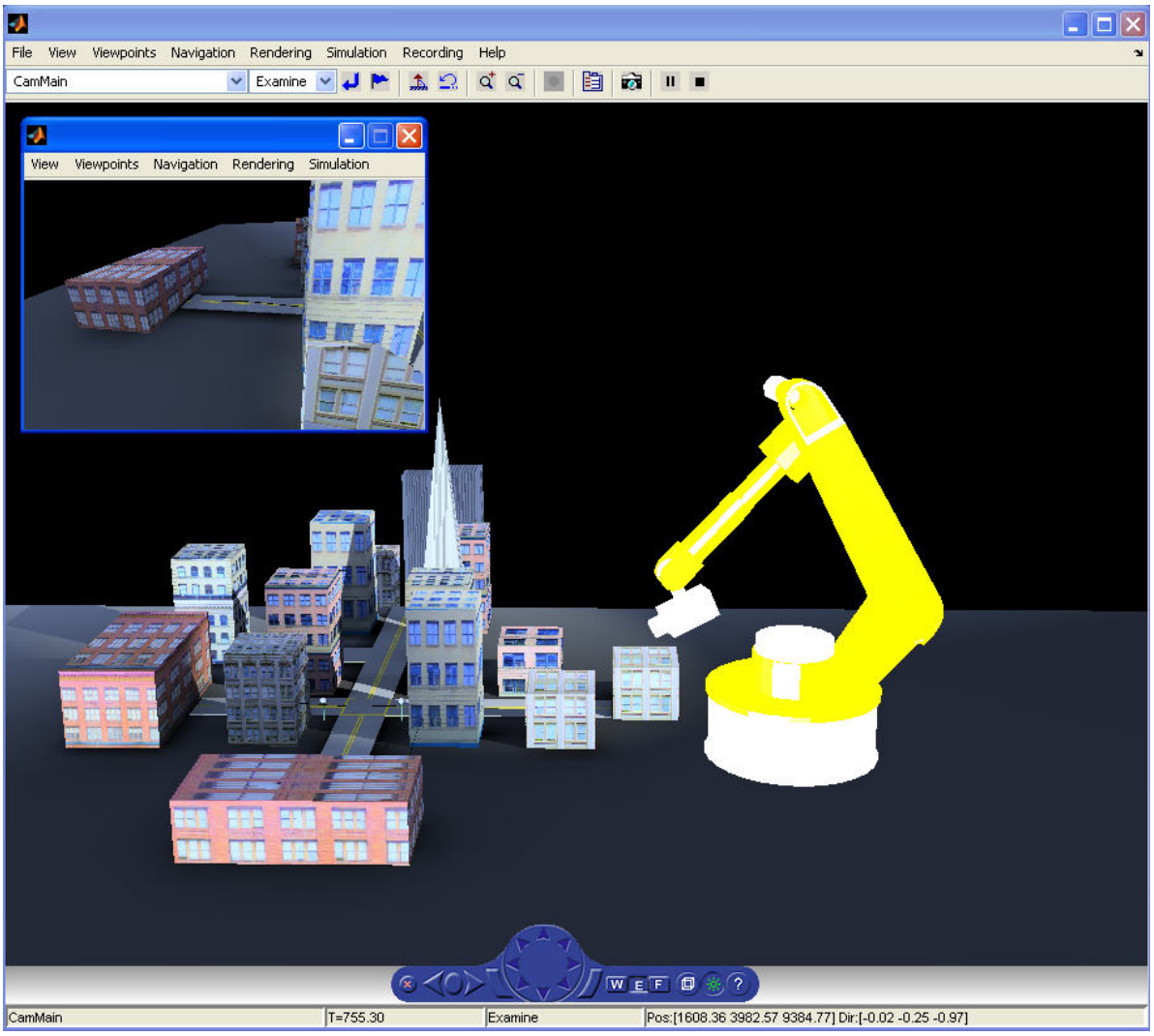


Figure 4.5.:Virtual robot in the virtual world [1]

camera in the global coordinate system, we must give and change the position of camera using the 3D mouse, this means we already know the position of the camera in the global coordinate system. Then we must compute six rotation angles of each axis to achieve the motion of camera. The condition of our project is the same as inverse kinematics. This means we can use inverse kinematics to solve our problem.

First we let every center of joint to be the origin of coordinate system and six links twist angles to be zero, then established the base coordinate system and another local coordinate system for

[1]This virtual robot is from the demo of Matlab 2009b/SimMechanics [23]; the virtual world [30].

every link $\{O_i; e_{i,1}, e_{i,2}, e_{i,3}\}$, witch is right-handed Cartesian coordinate system (see Fig. 4.6). $e_{i,1}$ is a direction vector of x_i and has the same direction as the rotation axis g_i. $e_{i,3}$ is a direction vector of z_i and has the same direction as the common normal between g_i and g_{i+1}. If there is no unique common normal, we can establish any coordinate system. From those conditions we established coordinate systems of our robot (see Fig. 4.6).

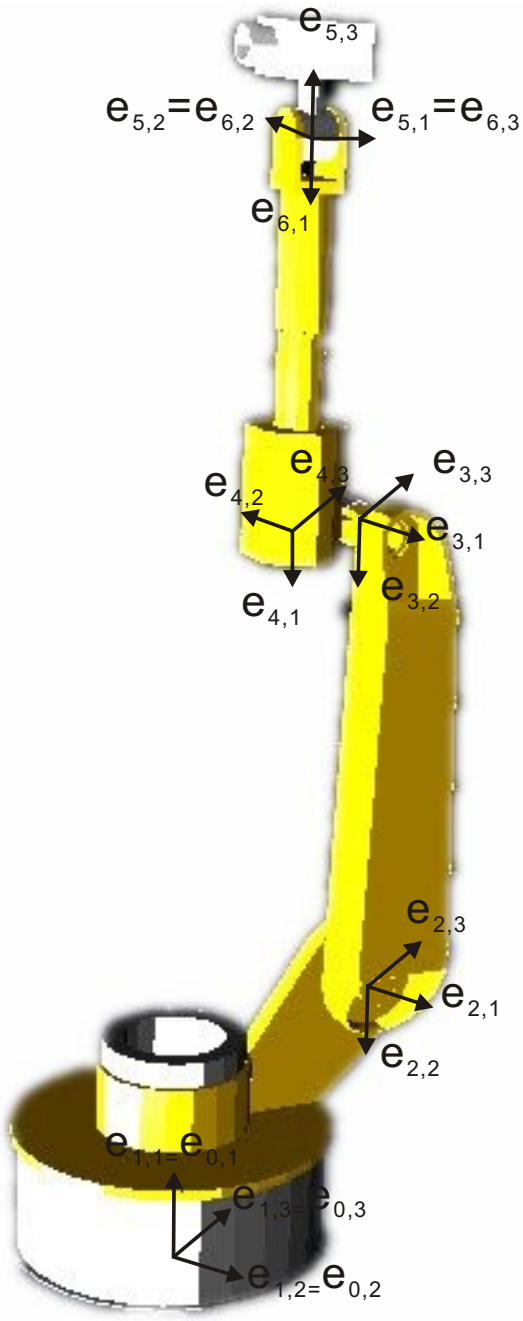


Figure 4.6.:Coordinate systems of our serial robot [1]

[1]This robot arm with virtual reality scene is from the demo of Matlab 2009b/SimMechanics [23].

We know $g_1 ... g_6$ as six axes of the serial robot. g_4, g_5, g_6 intersect in one point [11], so we split this problem into two steps. Firstly, we computed three angles of rotation u_1, u_2, u_3, then we computed the last angles of rotation u_4, u_5, u_6 [11].

4.3.1. Computing Angles of Rotation u_1, u_2, u_3

First of all, we looked at a point $P = [x_{i+1}, y_{i+1}, z_{i+1}]$, witch was based on $(i+1)^{st}$ coordinate system, and the same point $P = [x_i, y_i, z_i]$, which was based on i^{th} coordinate system. The homogeneous transformation for example during translation along x_{i+1}-axis and z_{i+1}-axis, rotation about x_{i+1}-axis and z_{i+1}-axis of the $(i+1)^{th}$ coordinate system to the i^{th} coordinate system is:

$$\begin{bmatrix} 1 \\ x_i \\ y_i \\ z_i \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ d_i & \cos(\alpha_i) & -\sin(\alpha_i) & 0 \\ 0 & \sin(\alpha_i) & \cos(\alpha_i) & 0 \\ a_i & 0 & 0 & 1 \end{bmatrix}}_{C_i} \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \cos(u_{i+1}) & -\sin(u_{i+1}) \\ 0 & 0 & \sin(u_{i+1}) & \cos(u_{i+1}) \end{bmatrix}}_{R_x(u_{i+1})} \begin{bmatrix} 1 \\ x_{i+1} \\ y_{i+1} \\ z_{i+1} \end{bmatrix} \tag{4.16}$$

The homogeneous transformation from the first to the zeroth coordinate system is only a rotation about the $e_{1,1}$ axis, since both coordinate system have a common x-axis:

$$\begin{bmatrix} 1 \\ x_0 \\ y_0 \\ z_0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \cos(u_1) & -\sin(u_1) \\ 0 & 0 & \sin(u_1) & \cos(u_1) \end{bmatrix} \begin{bmatrix} 1 \\ x_1 \\ y_1 \\ z_1 \end{bmatrix} \tag{4.17}$$

The homogeneous transformation from the second to the first coordinate system is a rotation about $e_{2,1}$ and $e_{2,3}$, and a translation along $e_{2,2}$, $e_{2,1}$ and $e_{2,3}$:

$$\begin{bmatrix} 1 \\ x_1 \\ y_1 \\ z_1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ d_1 & \cos(\alpha_1) & -\sin(\alpha_1) & 0 \\ a_1' & \sin(\alpha_1) & \cos(\alpha_1) & 0 \\ a_1 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \cos(u_2) & -\sin(u_2) \\ 0 & 0 & \sin(u_2) & \cos(u_2) \end{bmatrix} \begin{bmatrix} 1 \\ x_2 \\ y_2 \\ z_2 \end{bmatrix} \tag{4.18}$$

The homogeneous transformation from the third to the second coordinate system is a translation along $e_{3,2}$ and $e_{3,1}$:

$$\begin{bmatrix} 1 \\ x_2 \\ y_2 \\ z_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ d_2 & \cos(\alpha_2) & -\sin(\alpha_2) & 0 \\ a_2 & \sin(\alpha_2) & \cos(\alpha_2) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \cos(u_3) & -\sin(u_3) \\ 0 & 0 & \sin(u_3) & \cos(u_3) \end{bmatrix} \begin{bmatrix} 1 \\ x_3 \\ y_3 \\ z_3 \end{bmatrix} \tag{4.19}$$

The homogeneous transformation from the fourth to the third coordinate system is a rotation about $e_{4,3}$ and $e_{4,1}$, and a translation along $e_{4,3}$:

$$\begin{bmatrix} 1 \\ x_3 \\ y_3 \\ z_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ d_3 & \cos(\alpha_3) & -\sin(\alpha_3) & 0 \\ a_3 & \sin(\alpha_3) & \cos(\alpha_3) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \cos(u_4) & -\sin(u_4) \\ 0 & 0 & \sin(u_4) & \cos(u_4) \end{bmatrix} \begin{bmatrix} 1 \\ x_4 \\ y_4 \\ z_4 \end{bmatrix} \tag{4.20}$$

The homogeneous transformation from the fifth to the fourth coordinate system is a rotation about $e_{5,2}$ and $e_{5,1}$, and a translation along $e_{5,1}$:

$$\begin{bmatrix} 1 \\ x_4 \\ y_4 \\ z_4 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ d_4 & \cos(\alpha_4) & 0 & -\sin(\alpha_4) \\ 0 & 0 & 1 & 0 \\ a_4 & \sin(\alpha_4) & 0 & \cos(\alpha_4) \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \cos(u_5) & -\sin(u_5) \\ 0 & 0 & \sin(u_5) & \cos(u_5) \end{bmatrix} \begin{bmatrix} 1 \\ x_5 \\ y_5 \\ z_5 \end{bmatrix} \tag{4.21}$$

The homogeneous transformation from the sixth to the fifth coordinate system is a rotation about $e_{6,2}$ and $e_{6,1}$:

$$\begin{bmatrix} 1 \\ x_5 \\ y_5 \\ z_5 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ d_5 & \cos(\alpha_5) & 0 & -\sin(\alpha_5) \\ 0 & 0 & 1 & 0 \\ a_5 & \sin(\alpha_5) & 0 & \cos(\alpha_5) \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \cos(u_6) & -\sin(u_6) \\ 0 & 0 & \sin(u_6) & \cos(u_6) \end{bmatrix} \begin{bmatrix} 1 \\ x_6 \\ y_6 \\ z_6 \end{bmatrix} \tag{4.22}$$

We computed the location of the point $P = [x_0, y_0, z_0]$ of the end coordinate system in the global coordinate system from all homogeneous transformations:

$$\begin{bmatrix} 1 \\ x_0 \\ y_0 \\ z_0 \end{bmatrix} = R_x(u_1)C_1R_x(u_2)C_2R_x(u_3)C_3R_x(u_4)C_4R_x(u_5)C_5R_x(u_6) \begin{bmatrix} 1 \\ x_6 \\ y_6 \\ z_6 \end{bmatrix} \tag{4.23}$$

From equation 4.11 in section Inverse Kinematics and equation 4.15 in section 4.2 we can get the following equation:

$$B(u_1...u_6) = R_x(u_1)C_1R_x(u_2)C_2R_x(u_3)C_3R_x(u_4)C_4R_x(u_5)C_5R_x(u_6) \tag{4.24}$$

and

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ b_{10} & b_{11} & b_{12} & b_{31} \\ b_{20} & b_{21} & b_{22} & b_{32} \\ b_{30} & b_{31} & b_{23} & b_{33} \end{bmatrix} = R_x(u_1)C_1R_x(u_2)C_2R_x(u_3)C_3R_x(u_4)C_4R_x(u_5)C_5R_x(u_6) \tag{4.25}$$

The meaning of matrix equation 4.25 was already described in section 4.2.2. The coordinates of the origin O_6 are $[1, b_{10}, b_{20}, b_{30}]^T$, the coordinates of the end joint in the end coordinate system are $[1, 0, 0, 0]^T$. Now, we substitute these coordinates in equation 4.23:

$$\begin{bmatrix} 1 \\ x_0 \\ y_0 \\ z_0 \end{bmatrix} = R_x(u_1)C_1R_x(u_2)C_2R_x(u_3)C_3R_x(u_4)C_4R_x(u_5)C_5R_x(u_6) \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{4.26}$$

Then we get the following equation:

$$\begin{bmatrix} 1 \\ x_0 \\ y_0 \\ z_0 \end{bmatrix} = \begin{bmatrix} 1 \\ \mathrm{d_1{+}a_2cos(u_2) - a_3sin(u_2+u_3) + d_4cos(u_2+u_3)} \\ \mathrm{-(d_3{-}49.43)cos(u_1) - sin(u_1)(a_1{-}a_2sin(u_2) - a_3cos(u_2+u_3) - d_4sin(u_2+u_3))} \\ \mathrm{-(d_3{-}49.43) * sin(u_1) + cos(u_1)(a_1{-}a_2sin(u_2) - a_3cos(u_2+u_3) - d_4sin(u_2+u_3))} \end{bmatrix} \tag{4.27}$$

Since

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ b_{10} & b_{11} & b_{12} & b_{31} \\ b_{20} & b_{21} & b_{22} & b_{32} \\ b_{30} & b_{31} & b_{23} & b_{33} \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ b_{10} \\ b_{20} \\ b_{30} \end{bmatrix} \tag{4.28}$$

we get the following equations:

$$\mathrm{b_{10} = d_1{+}a_2cos(u_2) - a_3sin(u_2+u_3) + d_4cos(u_2+u_3)} \tag{4.29}$$

$$b_{20} = \mathrm{-(d_3{-}49.43)cos(u_1) - sin(u_1)(a_1{-}a_2sin(u_2) - a_3cos(u_2+u_3) - d_4sin(u_2+u_3))} \tag{4.30}$$

$$\mathrm{b_{30} = -(d_3{-}49.43)sin(u_1) + cos(u_1)(a_1{-}a_2sin(u_2) - a_3cos(u_2+u_3) - d_4sin(u_2+u_3))} \tag{4.31}$$

First, we computed u_1 from equation 4.30 and 4.31, from equation 4.31 we got the equation:

$$\mathrm{a_1{-}a_2sin(u_2) - a_3cos(u_2+u_3) - d_4sin(u_2+u_3) = P = \frac{b_{30}{+}(d_3{-}49.43)sin(u_1)}{cos(u_1)}} \tag{4.32}$$

Then we substituted P in equation 4.30 and got the following equation:

$$\mathrm{b_{20}cos(u_1){+}b_{30}sin(u_1) = -(d_3{-}49.43);} \tag{4.33}$$

Because of following equation:

$$\mathrm{acos(x) + bsin(x) = \sqrt{a^2 + b^2}cos(x+a); tan(a) = \frac{b}{a}} \tag{4.34}$$

we got an equation about u_1:

$$\mathrm{u_1 = arccos(\frac{-(d_3{-}49.43)}{\sqrt{b_{30}^2 + b_{20}^2}}) + arctan(\frac{b_{30}}{b_{20}});} \tag{4.35}$$

We got two solutions $u_{1,1}$ and $u_{1,2}$ from equation 4.35 in interval $[0, 2\pi)$. If $b_{20} > 0$ we can take solution $u_{1,1}$, if $b_{20} < 0$ we get the solution $u_{1,2} = u_{1,1} + \pi$, if $b_{20} = 0$ and $b_{30} < 0$, $\tan(\mathrm{a}) = \frac{\mathrm{b}}{\mathrm{a}} = -\frac{\pi}{2}$, if $b_{20} = 0$ and $b_{30} > 0$, $\tan(\mathrm{a}) = \frac{\mathrm{b}}{\mathrm{a}} = \frac{\pi}{2}$. There is an other situation of u_1, if $\frac{-(\mathrm{d}_3 - 49.43)}{\sqrt{\mathrm{b}_{30}^2 + \mathrm{b}_{20}^2}} > 1$ the part of $\cos(u_1)$ has no solution, we let $u_1 = \arctan(\frac{b_{30}}{b_{20}})$.

Then we computed u_3 from equation 4.29, 4.30 and 4.31, we considered the position $b_{30} = 0$ and $b_{20} = 0$, then we used the following equation to solve this problem:

$$v = \begin{cases} \frac{b_{20} + (\mathrm{d}_3 - 49.43)\sin(\mathrm{u}_1)}{-\sin(\mathrm{u}_1)} - \mathrm{a}_1 & (b_{20} \neq 0) \\ \frac{b_{30} + (\mathrm{d}_3 - 49.43)\sin(\mathrm{u}_1)}{\cos(\mathrm{u}_1)} - \mathrm{a}_1 & (b_{20} = 0) \end{cases} \tag{4.36}$$

From equation 4.36 we got a new equation to substitute equation 4.30 and 4.31:

$$-\mathrm{a}_2\sin(\mathrm{u}_2) - \mathrm{a}_3\cos(\mathrm{u}_2 + \mathrm{u}_3) - \mathrm{d}_4\sin(\mathrm{u}_2 + \mathrm{u}_3) = v \tag{4.37}$$

Then u_3 was computed from equation 4.29 and 4.37, these two equations have the same terms of trigonometric functions but different coefficients, so we squared and added them, finally we got the following equation:

This equation was the same as equation 4.34, so u_3 can be computed:

$$\mathrm{u}_3 = \arccos(\frac{\mathrm{v}^2 + (b_{10} - d_1)^2 - (d_4^2 + a_2^2 + a_3^2)}{\sqrt{(2a_2d_4)^2 + (2a_2a_3)^2}}) + \arctan(\frac{-(2a_2d_4)}{(2a_2a_3)}); \tag{4.38}$$

However u_3 has two solutions, but only the solution in the interval $[0, \pi]$ is meaningful for our project. There is also a different situation, if $\frac{\mathrm{v}^2 + (b_{10} - d_1)^2 - (d_4^2 + a_2^2 + a_3^2)}{\sqrt{(2a_2d_4)^2 + (2a_2a_3)^2}} > 1$, we let $\mathrm{u}_3 = \arctan(\frac{-(2a_2d_4)}{(2a_2a_3)})$. we already knew the solution of u_3 and u_1, then we computed u_2 from equation 4.29 and 4.37. They had to be resolved, then we got following equation :

$$b_{10} - d_1 = \cos(\mathrm{u}_2)(\mathrm{a}_2 - \mathrm{a}_3\sin(\mathrm{u}_3) + \mathrm{d}_4\cos(\mathrm{u}_3)) - \sin(\mathrm{u}_2)(\mathrm{a}_3\cos(\mathrm{u}_3) + \mathrm{d}_4\sin(\mathrm{u}_3)) \tag{4.39}$$

$$v = -\sin(\mathrm{u}_2)(\mathrm{a}_2 - \mathrm{a}_3\sin(\mathrm{u}_3) + \mathrm{d}_4\cos(\mathrm{u}_3)) - \cos(\mathrm{u}_2)(\mathrm{a}_3\cos(\mathrm{u}_3) + \mathrm{d}_4\sin(\mathrm{u}_3)) \tag{4.40}$$

We got two equations for u_1 from these equations:

$$\sin(\mathrm{u}_2) := -(b_{10} - d_1 - \cos(\mathrm{u}_2)(\mathrm{a}_2 - \mathrm{a}_3\sin(\mathrm{u}_3) + \mathrm{d}_4\cos(\mathrm{u}_3)))/(\mathrm{a}_3\cos(\mathrm{u}_3) + \mathrm{d}_4\sin(\mathrm{u}_3)) \tag{4.41}$$

$$\cos(\mathrm{u}_2) := (b_{10} - d_1 + \sin(\mathrm{u}_2)(\mathrm{a}_3\cos(\mathrm{u}_3) + \mathrm{d}_4\sin(\mathrm{u}_3)))/(\mathrm{a}_2 - \mathrm{a}_3 * \sin(\mathrm{u}_3) + \mathrm{d}_4\cos(\mathrm{u}_3)) \tag{4.42}$$

u_2 can be computed with the arctangent function, the solution of u_2 is only meaningful in interval $[0, \pi]$. For us, in the situation of $(b_{10} - d_1 + \sin(\mathrm{u}_2)(\mathrm{a}_3\cos(\mathrm{u}_3) + \mathrm{d}_4\sin(\mathrm{u}_3))) = 0$, this means the value of arctangent is endless great, then $u_2 = \frac{\pi}{2}$.

4.3.2. Computing Angles of Rotation u_4, u_5, u_6

The origin of the end coordinate system is yielded during calculation of u_1,u_2 and u_3 in the global coordinate system. Then the direction of $e_{6,1}$, $e_{6,2}$ and $e_{6,3}$ was given by the 3D mouse, which ascertain the direction of the camera. u_4, u_5, u_6 are computed during $e_{6,1}$, $e_{6,2}$ and $e_{6,3}$ and also the other direction vectors. u_4 is only dependent on direction vector $e_{5,1}$ and $e_{4,3}$, when $u_4 = 0$ $e_{5,1} = e_{4,3}$. The direction of $e_{4,1}$, $e_{4,2}$, $e_{4,3}$ are only dependent on rotation about g_1, g_2, g_3, this means the direction of axes of the fourth coordinate system is only dependent on u_1, u_2 and u_3, we can get direction vector of the fourth coordinate system from equation 4.23 and 4.25, first the matrix of homogeneous transformation from zeroth to fourth is computed with following equation:

$$B(u_1...u_3) = R_x(u_1)C_1R_x(u_2)C_2R_x(u_3)C_3 \tag{4.43}$$

$$\begin{bmatrix} 0 & 1 & 0 & 0 \end{bmatrix}^T$$

Because of equation 4.23, 4.25 and 4.14, we got direction vector of $e_{4,1}$ and $e_{4,3}$ by multiplying $B(u_1...u_3)$ with $\begin{bmatrix} 0 & 1 & 0 & 0 \end{bmatrix}^T$ and $\begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix}^T$:

$$e_{4,1} = \begin{bmatrix} -\cos(u_2 + u_3) \\ -\sin(u1)\sin(u_2 + u_3) \\ \cos(u_1)\sin(u_2 + u_3) \end{bmatrix} \tag{4.44}$$

$$e_{4,3} = \begin{bmatrix} \sin(u_2 + u_3) \\ -\sin(u1)\cos(u_2 + u_3) \\ \cos(u_1)\cos(u_2 + u_3) \end{bmatrix} \tag{4.45}$$

$e_{5,1}$ is dependent on $e_{4,1}$ and $e_{6,1}$, with cross product of $e_{4,1}$ and $e_{6,1}$ is $e_{5,1}$ computed:

$$e_{4,1} \times e_{6,1} \tag{4.46}$$

Since the cross product is not a unit vector, this means the length of the cross product is not one, it must be translated to a unit vector with following equation:

$$e_{5,1} = \frac{e_{4,1} \times e_{6,1}}{|e_{4,1} \times e_{6,1}|} \tag{4.47}$$

We must consider that, if cross product of $e_{4,1}$ and $e_{6,1}$ is zero, $e_{5,1}$ is the same as $e_{4,3}$ or -$e_{4,3}$, we let $e_{5,1} = e_{4,3}$.

From equation 3.5 in chapter 3, we can compute the angle between two direction vector, because the direction vector is a unit vector, so we can get following equation with unit vectors $\vec{a}$ and $\vec{b}$:

$$\cos(\theta) = \vec{a} \cdot \vec{b} \tag{4.48}$$

Then u_4 is computed from $e_{5,1}$ and $e_{4,3}$:

$$u_4 = \arccos(e_{4,3} \cdot e_{5,1}) \tag{4.49}$$

There is a special situation, if direction vectors are collinear, their cross product is zero. There are two possibilities: These two direction vectors have the same or the reverse direction. If we known the direction of both vectors, $u_4 = 0$ or π can be ascertained.

During tests of this method we found that, that the solution of equation 4.48 was in interval $[0, \pi]$, but the solution, what we needed was in interval $[0, 2\pi)$. To solve this problem we needed to know the direction of axis of rotation. The angle between two vectors could be said as an angle of rotation about the axis, which is the cross product of them, from one vector to the other. The cross product of two vectors is defined as a vector that is perpendicular to both vectors, with a direction given by the right-hand rule and a magnitude equal to the area of the parallelogram that the vectors span (see Fig. 4.7).

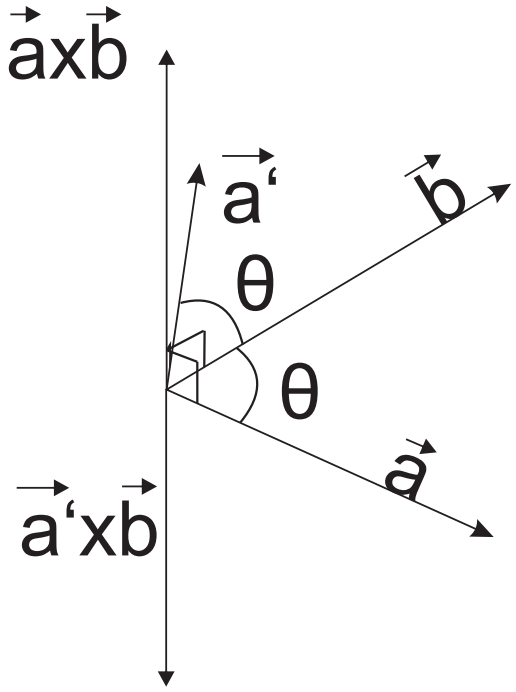


Figure 4.7.:Cross product of the vector

From Fig. 4.7 we know that the direction of rotation axis is different, if the rotation angle is in different intervals, this means the direction of rotation axis for rotation angle in interval $[0, \pi]$ is in the reverse direction of rotation axis for rotation angle in interval $(\pi, 2\pi)$. The interval of rotation angle ascertained by convention, in the way, that the rotation angle counterclockwise about robot axis of ith coordinate system is positive. So that if direction of rotation axis is the same as direction of $e_{i,1}$, rotation angle is in interval [0,π], otherwise in interval $(\pi, 2\pi)$. Rotation angle can be computed in interval $[0, 2\pi)$ with the following equation: The rotation axis and $e_{4,1}$ are collinear, if direction of rotation axis is the same as direction of $e_{4,1}$

$$u_4 = \arccos(e_{4,3} \cdot e_{5,1}) \tag{4.50}$$

Otherwise:

$$u_4 = 2\pi - \arccos(e_{4,3} \cdot e_{5,1}) \tag{4.51}$$

Because of $e_{4,1}$, $e_{5,1}$ and $e_{6,1}$ cross in one point, so the angle between $e_{4,1}$ and $e_{6,1}$ is only influenced by rotation about $e_{5,1}$, so that u_5 can be computed from $e_{4,1}$ and $e_{6,1}$: The rotation axis and $e_{5,1}$ are collinear, if direction of rotation axis is the same as direction of $e_{5,1}$

$$u_5 = \arccos(e_{4,1} \cdot e_{6,1}) \tag{4.52}$$

Otherwise:

$$u_5 = 2\pi - \arccos(e_{4,1} \cdot e_{6,1}) \tag{4.53}$$

The angle between $e_{5,1}$ and $e_{6,3}$ is only influenced by rotation about $e_{6,1}$, so that u_6 can be computed from $e_{5,1}$ and $e_{6,3}$: The rotation axis and $e_{6,1}$ are collinear, if direction of rotation axis is the same as direction of $e_{6,1}$

$$u_6 = \arccos(e_{5,1} \cdot e_{6,3}) \tag{4.54}$$

Otherwise:

$$u_6 = 2\pi - \arccos(e_{5,1} \cdot e_{6,3}) \tag{4.55}$$

We wrote all equations, which can compute all rotation angles of the robot axes, with Embedded Matlab Function in Matlab/Simulink. The Matlab code was written in listing A.1. Because of $e_{6,1}$ and $e_{6,3}$ must be calculated, and we got the initial values of $e_{6,1}$ and $e_{6,3}$ at the initial situation of the robot (all rotation angles are zero). The direction of the camera is controlled by 3D mouse with following equation:

$$\mathrm{e}_{6,1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(w2) & -\sin(w2) \\ 0 & \sin(w2) & \cos(w2) \end{bmatrix} \begin{bmatrix} \cos(w1) & 0 & \sin(w1) \\ 0 & 1 & 0 \\ -\sin(w1) & 0 & \cos(w1) \end{bmatrix} \begin{bmatrix} -1 \\ 0 \\ 0 \end{bmatrix} \tag{4.56}$$

$w1$ and $w2$ are angles, which are given by the 3D mouse, using this equation the camera is rotated about x and y axis of the global coordinate system.

$e_{6,3}$ is dependent on $e_{6,1}$, which means the direction of $e_{6,3}$ is changed by rotation of $e_{6,1}$ and direction of $e_{6,1}$, this relation is described by the following equation:

$$\mathrm{e}_{6,3} = \begin{bmatrix} \mathrm{e}_{6,1}(1)^2(1-\mathrm{c(w3)}) + \mathrm{c(w3)} & \mathrm{e}_{6,1}(1)\mathrm{e}_{6,1}(2)(1-\mathrm{c(w3)}) - \mathrm{e}_{6,1}(3)\mathrm{s(w3)} & \mathrm{e}_{6,1}(1)\mathrm{e}_{6,1}(3)(1-\mathrm{c(w3)}) + \mathrm{e}_{6,1}(2)\mathrm{s(w3)} \\ \mathrm{e}_{6,1}(1)\mathrm{e}_{6,1}(2)(1-\mathrm{c(w3)}) + \mathrm{e}_{6,1}(3)\mathrm{s(w3)} & \mathrm{e}_{6,1}(2)^2(1-\mathrm{c(w3)}) + \mathrm{c(w3)} & \mathrm{e}_{6,1}(2)\mathrm{e}_{6,1}(3)(1-\mathrm{c(w3)}) - \mathrm{e}_{6,1}(1)\mathrm{s(w3)} \\ \mathrm{e}_{6,1}(1)\mathrm{e}_{6,1}(3)(1-\mathrm{c(w3)}) - \mathrm{e}_{6,1}(2)\mathrm{s(w3)} & \mathrm{e}_{6,1}(2)\mathrm{e}_{6,1}(3)(1-\mathrm{c(w3)}) + \mathrm{e}_{6,1}(1)\mathrm{s(w3)} & \mathrm{e}_{6,1}(3)^2(1-\mathrm{c(w3)}) + \mathrm{c(w3)} \end{bmatrix} \\ \begin{bmatrix} 1 & 0 & 0 \\ 0 & \mathrm{c}(w2) & -\mathrm{s}(w2) \\ 0 & \mathrm{s}(w2) & \mathrm{c}(w2) \end{bmatrix} \begin{bmatrix} \mathrm{c}(w1) & 0 & \mathrm{s}(w1) \\ 0 & 1 & 0 \\ -\mathrm{s}(w1) & 0 & \mathrm{c}(w1) \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \tag{4.57}$$

Where

c...cos;
s...sin;

$$\mathrm{e}_{6,1} = \begin{bmatrix} \mathrm{e}_{6,1}(1) \\ \mathrm{e}_{6,1}(2) \\ \mathrm{e}_{6,1}(3) \end{bmatrix} \tag{4.58}$$

and $w1$, $w2$, $w3$ are given by 3D mouse. We developed a Matlab code for this experiment (see Fig. 4.9).

First, we changed the position of camera accurately to know the accuracy of our Matlab code, so we used Slider Gain block, which vary a scalar gain during a simulation using a slider [22], to change the position of camera. We gave the rotation angle, which was computed by the Matlab code in Embedded MATLAB Function (see Fig. 4.10), to the port of each link (see Fig. 4.3) for our experiment.

We got the x, y and z values, which were computed from six rotation angles. They were the same as the value of inputs, which means our Matlab code is accurate and the result of our Matlab code is non-ambiguous. Then, we experimented on 3D mouse, which can control the coordinates of the origin of the end coordinate system in the global coordinate system and the directions of axes of the end coordinate system in the global coordinate system.

During our experiment, we got a problem, because of the Denavit-Hartenberg parameters of our virtual robot: $e_{1,1}$ and $e_{4,1}$ are not collinear, the length of the common normal from $e_{1,1}$ to $e_{4,1}$ is 156, this means there is a dead zone for our robot. There was a scale 2000 from real robot to our virtual robot, which means, the value from virtual world is the same as the value of the real world divided by 2000. Then we got the length of the common normal from $e_{1,1}$ to $e_{4,1}$ of $156/2000 = 0.078$. This means, the position of the origin of the end coordinate system is impossible in the region, which is a cylinder with a radius of 0.078 and the axis of this cylinder is $e_{0,1}$. So that y and z values of the position of the end coordinate system is limited by following equation:

$$y^2 + z^2 \geq 0.078^2 \tag{4.59}$$

Because of Denavit-Hartenberg parameters of our robot, the x, y and z values of the origin of the end coordinate system are also limited, the maximum value of x in the second coordinate system is computed from equation 4.3 and 4.7, which is $(4249 - 1012)/2000 = 1.6185$, and the maximum of y and z are limited by x, if $x = (1012 - 804.8 + 498.5)/2000$, y and z have the maximum value, if $x = 1.6185$, y and z have the minimum value. y and z values are limited by following equation:

$$z^2 + y^2 \leq 1.6185^2 - x^2 - \frac{498.5}{2000} \tag{4.60}$$

We developed a Matlab code for this equation (see Fig. 4.12), we used Embedded MATLAB Function3 to compute z^2 and y^2 and the If block together with If Action subsystems containing Action Port blocks to implement standard C-like if-else logic [22]. The If Action subsystem, which is a Subsystem block that is preconfigured to serve as a starting point for creating a subsystem whose execution is triggered by an If block [22], to stop and start the motion of the robot.

The rotation angle of g_6 (u_6) of our virtual robot is limited in interval $[-\frac{2\pi}{3}, \frac{2\pi}{3}]$, because we only study the motion of camera, we did not consider this limit of u_6 and the size of the camera.

Then we also studied the tracking of camera with our virtual robot like in chapter 3. We developed code (see Fig. 3.11) to compute the direction from camera to a point, the position of point was given as precondition and the position of camera was given by the 3D mouse. Then

we got a direction vector from origin of the end coordinate system to the point (see Fig. 4.13). We got a problem that we did not get a direction vector from camera to this point, because we built the camera on the robot axis g_6 but not on the origin of the end coordinate system. And the direction of camera is the same as the direction $e_{6,3}$, first, we let $e_{6,3}$ be the same as this direction vector, then the direction vector of $e_{6,1}$ must be the same as direction vector, which computed out from cross product of direct vector of camera and direction vector from origin of end coordinate system to the point. We got the camera tracked on the correct direction but not on the correct point, however we can change the tracking point around the correct point during rotation of camera about $e_{6,3}$ and the camera tracked on a fixed point during translation (see Fig. 4.5). If we built the camera on the robot axis g_6 and let direction of camera be the same as direction of $e_{6,1}$, further let the direction vector of $e_{6,1}$ be the same as direction vector from origin of the end coordinate system to the point, and finally let the direction vector of $e_{6,3}$ be the same as direction vector, which was computed from the cross product of the direction vector of the camera and direction vector from the origin of end coordinate system to that point, then the camera focused on the correct point during changing the position of camera (see Fig. 4.8).

Finally, the position and direction of camera in global coordinate system is well controlled with the 3D mouse by our programm, the position and the direction of the camera can be also changed in the own coordinate system of the end link.

[1]This virtual robot is from the demo of Matlab 2009b/SimMechanics [23]. Virtual world [30].

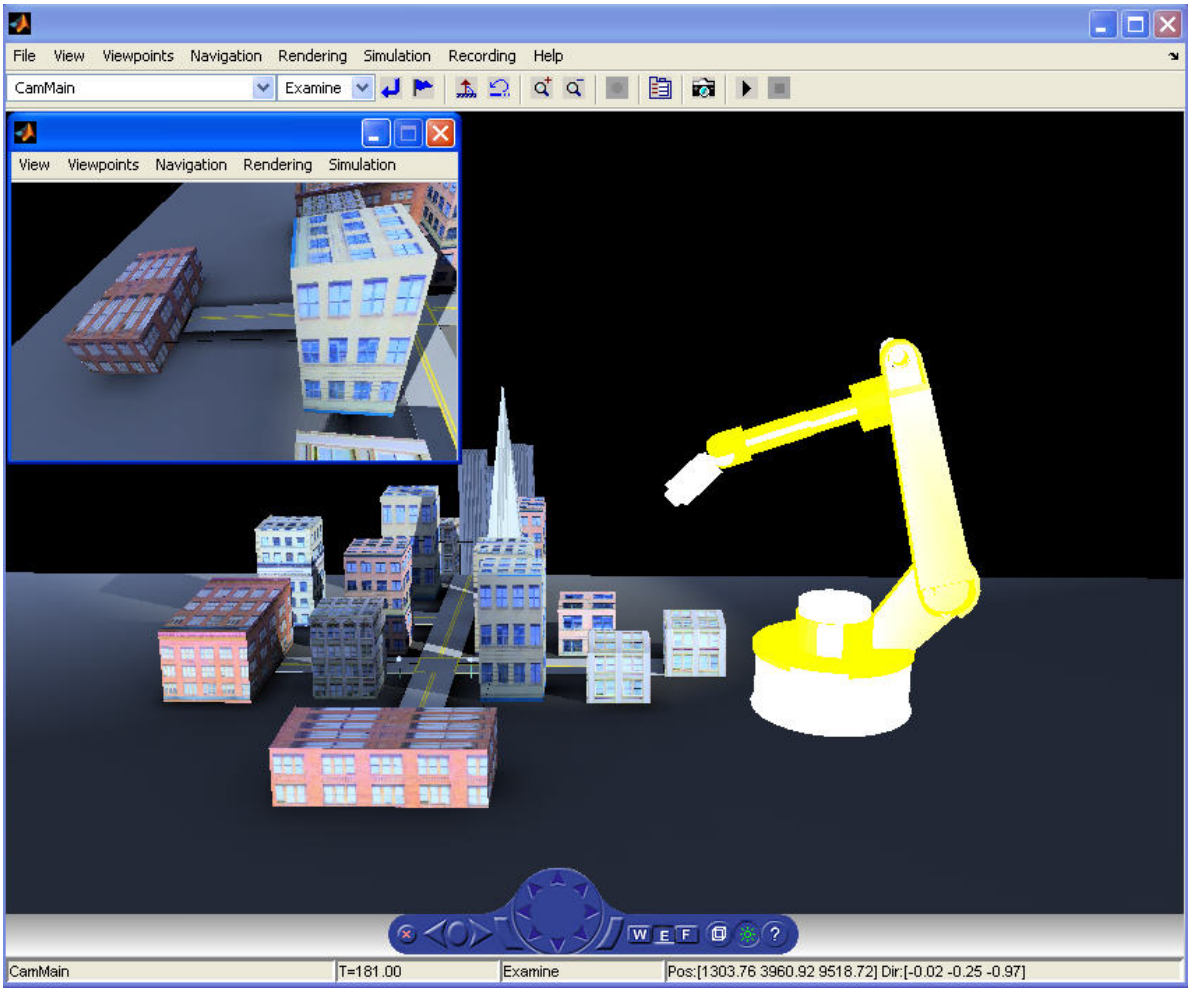


Figure 4.8.:Matlab code for tracking of camera on a correct point [1]

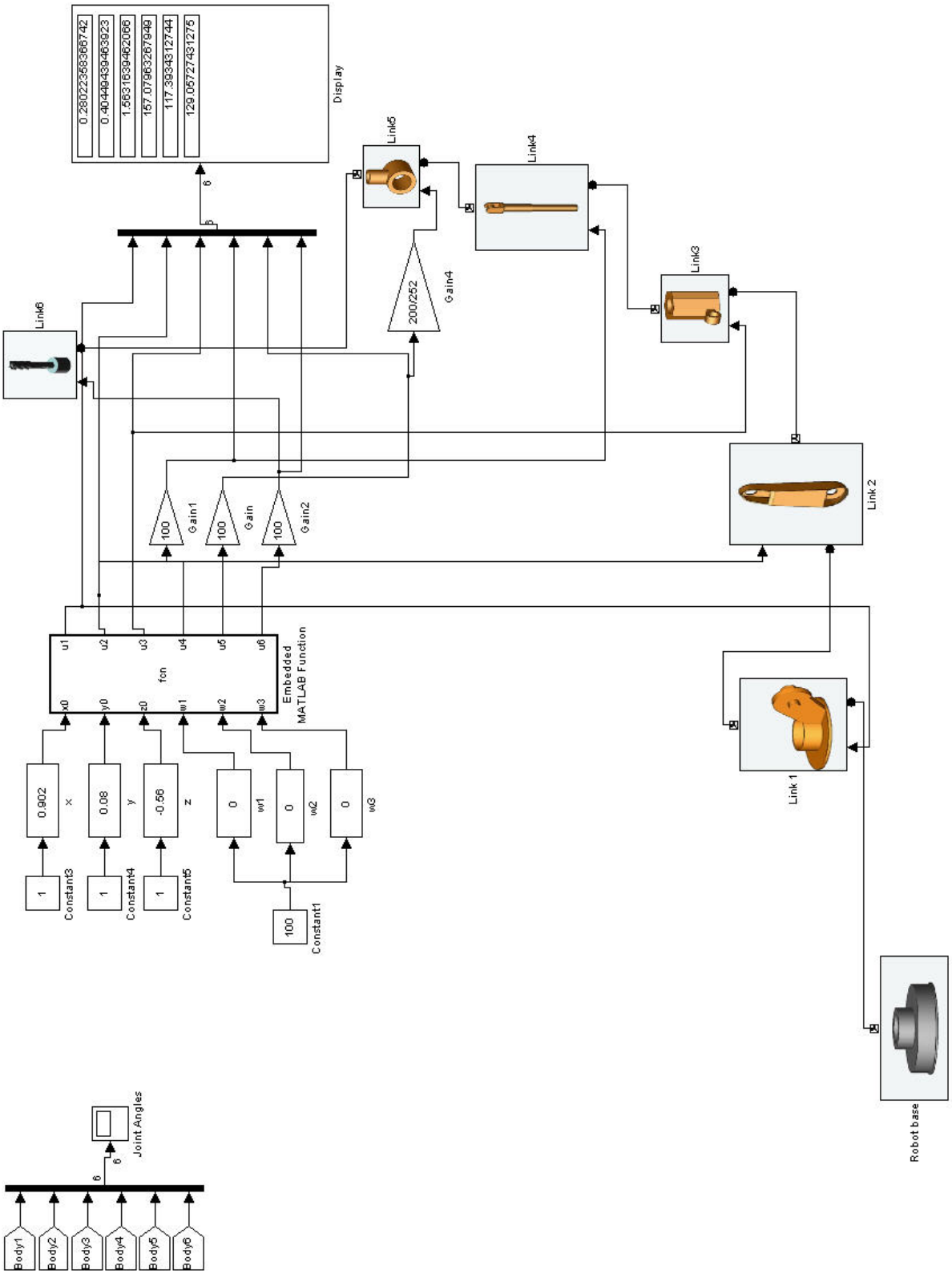


Figure 4.9.:Matlab code for the experiment on Slider Gain block

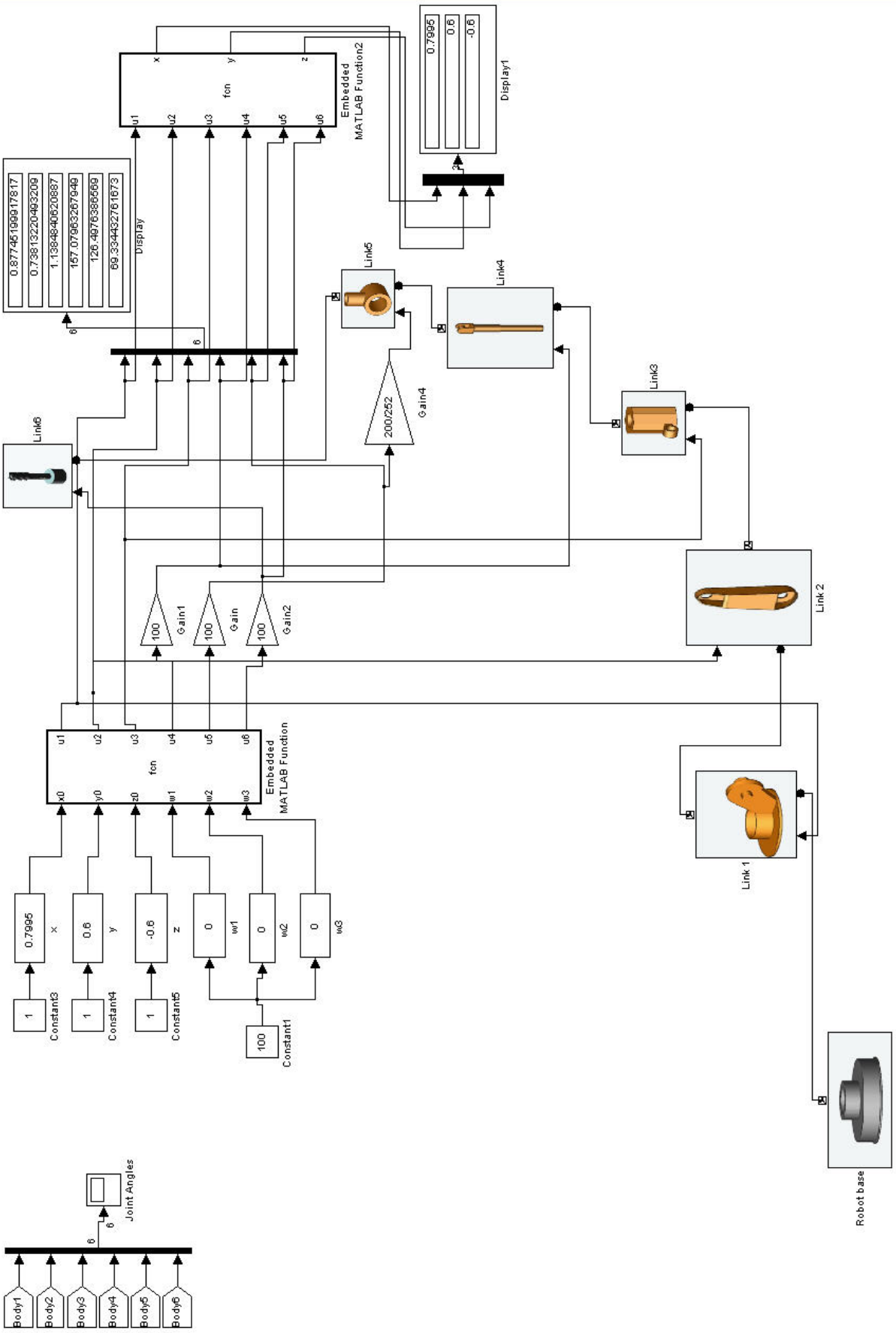


Figure 4.10.:Comparing the values of input with the values of output

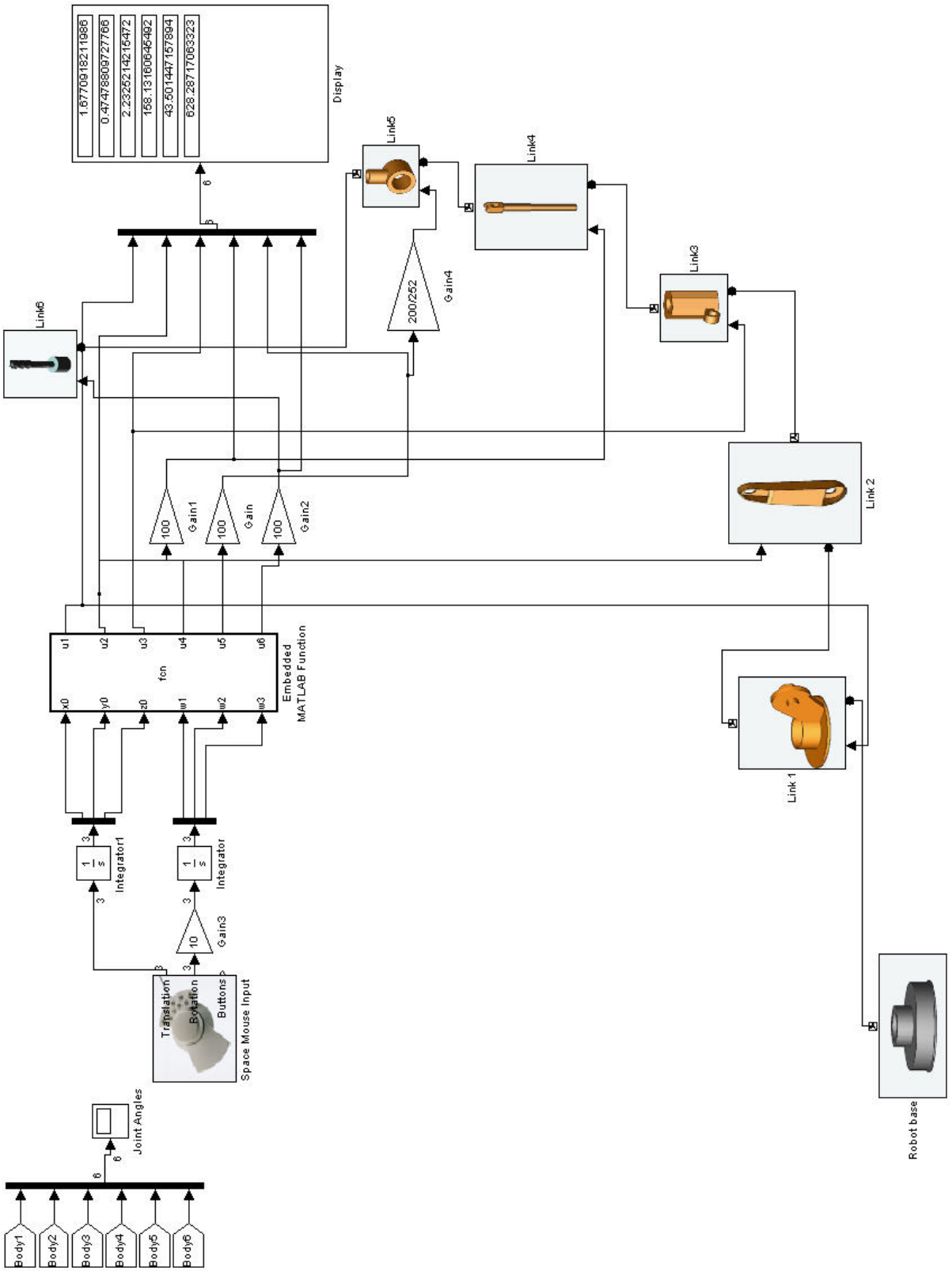


Figure 4.11.:Matlab code for the experiment with the 3D mouse

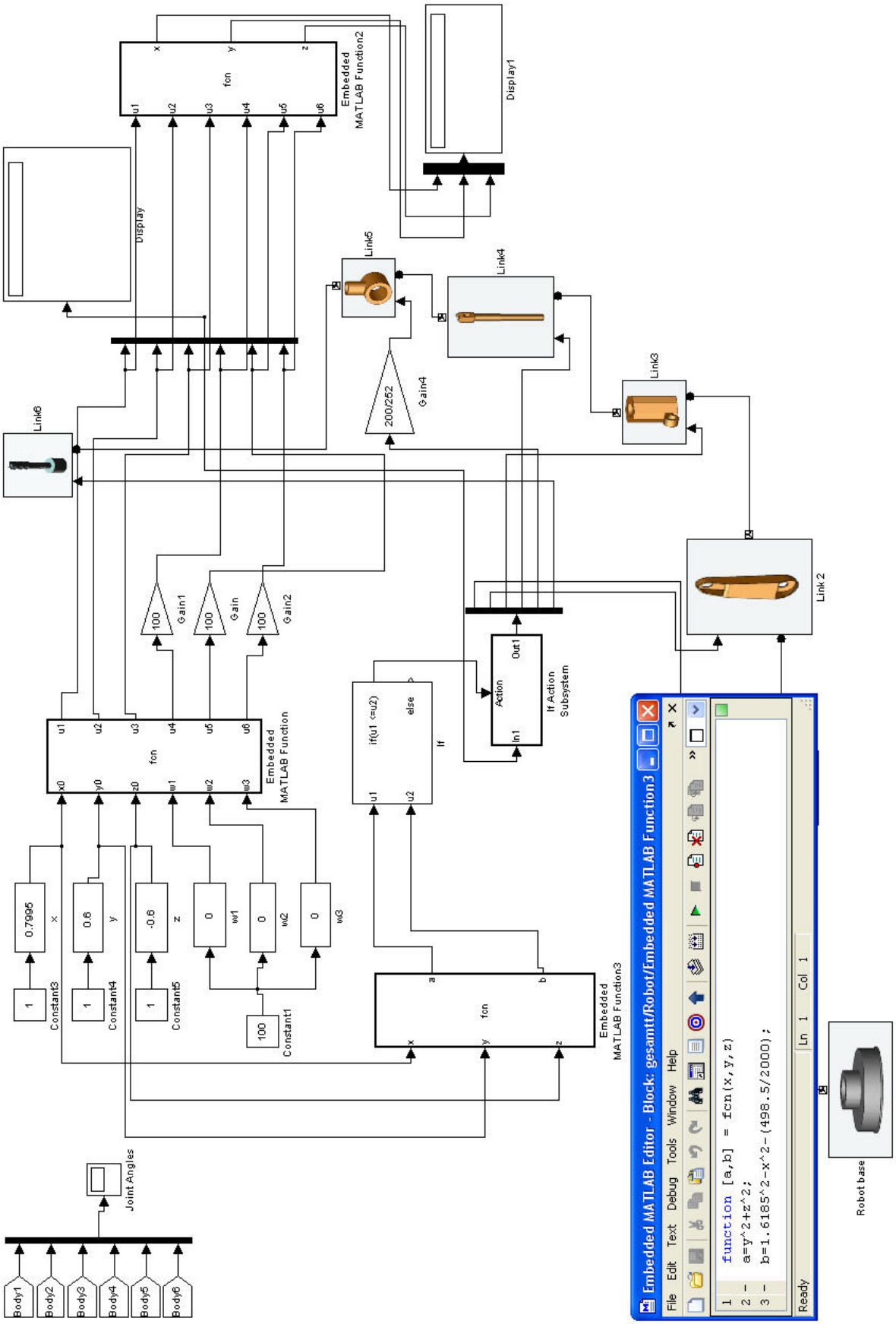


Figure 4.12.:Matlab code for limits of y and z coordinate values

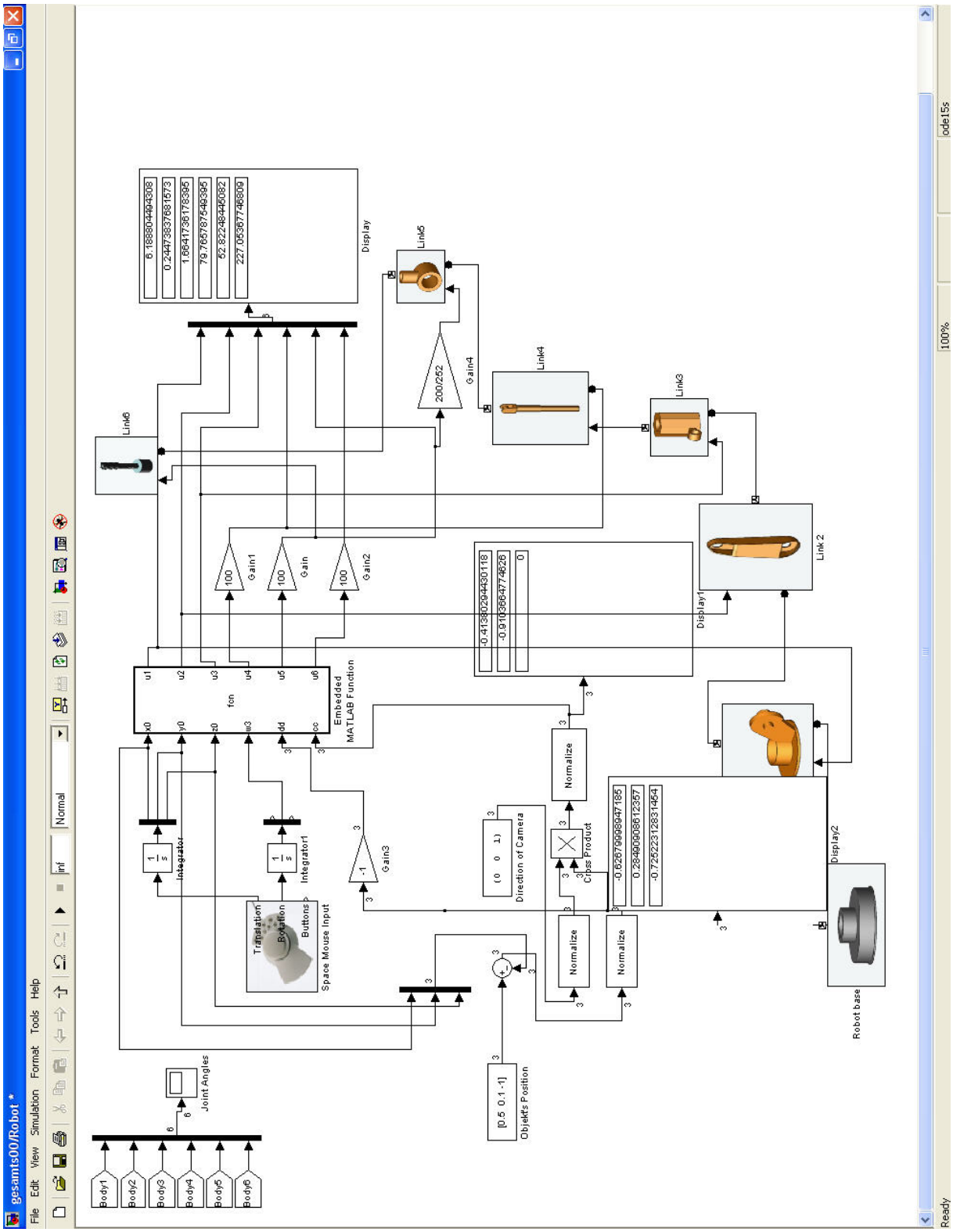


Figure 4.13.:Matlab code for tracking of camera on a point

5. Conclusion

This thesis introduced into mathematics and kinematics of the motion of a robot-driven camera and documented experiments with a 3D mouse for control. Camera and the 6DOF serial robot were built in a virtual world in VRML. The 3D mouse gave six signals to the outputs, three rotation signals and three translation signals, so that a 3D mouse was compatible for our study of motion in three dimension space. The virtual robot is a detailed dynamic model of the Manutec r3 robot. Our experimental results verify the mathematical analysis and make it applicable for implementation on a real robot.

In chapter 2 several mathematical methods to describe motion known from the literature were presented. The motion included translation and rotation. Some problems with rotation in three dimensions were discussed, for example the rotation matrix with Euler angles had the gimbal lock problem, which the quaternions had not. On the other hand, quaternions had problems with translation.

A Matlab code was developed to control the motion of the camera in VRML world with a 3D mouse. This Matlab code included mathematical equations, which converted outputs of 3D mouse to the camera motion in VRML world. A method was developed to solve the problem with gimbal lock and concatenation of translation and rotation, which used infinitesimal calculus to calculate direction vector of axes of coordinate frame, which was fixed on the moving camera. This method worked well to solve gimbal lock by rotation with Euler angles and translation problem of quaternions. Additionally, a Matlab code was developed to solve the problem to let the camera follow an object, when positions of the camera or the object are changing. This Matlab code worked well and moves the camera like human eyes, which can track resting or moving objects.

We also studied kinematics for the motion of the robot, which was used to move camera in our project. A Matlab code was developed to control the motion of camera, which was mounted on the robot, with a 3D mouse. This program worked well to control the position of the camera and had a non-ambiguous solution for each position, however the position of camera was limited by the physical contraints of camera and links. With this program the view direction of the camera can be kept during a translation of the camera.

The resulting software turned out to be suitable to navigate a camera with a 3D mouse and to follow the motion of an object in a virtual world. The method of this thesis is meaningful for 3D games, virtual movies and VRML environment. It will be useful to control real robots in production industry for image inspection.

A. Matlab Code

Listing A.1:Calculation of $u_1 ... u_6$ from $x0, y0, z0, w1, w2, w3$

```
1
2  function [u1,u2,u3,u4,u5,u6,e51,e63,e611,uu22] = fcn(x0,y0,z0,
       w1,w2,w3)
3  %
4  %  This Matlab code is for the control of direction and
       position of the camera, which is builded on the end effector
        of 6DOF robot, with a 3D mouse.
5  %
6  %  The inputs x0, y0 and z0 are x, y and z values of origin of
       the end coordinate system in the global coordinate system,
       which are gave by the 3D
7  %  mouse.
8  %  w1, w2, w3 are rotation angles, which are gave by the 3D
       mouse. The direction of axis of the end coordinate system is
        computed from rotation
9  %  angles.
10 %
11 %  The outputs from $u_1$ to $u_6$ are rotation angles of the
       robot joints, which are computed from x0, y0, z0 and w1, w2,
        w3. The outputs are used for %  control
12 %  of motion of robot links.
13 %
14 %  Author: Yuan Li
15 %  Date:   20 October 2010
16
17 %Denavit-Hartenberg parameters
18 %2000 is the scale from virtual robot to VRML-World
19 d1=1012/2000;
20 %d2=0;
21 d3=156/2000;
22 d4=1456/2000;
23 %d5=0;
24 a1=804.8/2000;
25 a2=1781/2000;
26 a3=306.3/2000;
27 %a4=0
28 R=-d3;
29
```

```
% First step:
% The rotation angle of e(1,1)
if(abs(R/sqrt(z0^2+y0^2))<=1)
 if(y0>0)
   u1=acos(R/sqrt(z0^2+y0^2))+atan(z0/y0);
 elseif(y0==0&&z0>0)
   u1=acos(R/sqrt(z0^2+y0^2))+pi/2;
 elseif(y0==0&&z0<0)
   u1=acos(R/sqrt(z0^2+y0^2))-pi/2;
 elseif (y0<0)
   u1=acos(R/sqrt(z0^2+y0^2))+atan(z0/y0)+pi;
 else
   u1=0;
 end
elseif(y0~=0)
    u1=pi/2;
else
    u1=0;
end

% 2. Step:
% The rotation angle u3 of e(3,1) is computed
if(y0~=0)
    c=-(y0+d3*cos(u1))/sin(u1)-a1;
else
    c=(z0+d3*sin(u1))/cos(u1)-a1;
end

b=-2*a2*a3;
a=2*a2*d4;
p1=x0-d1;

if(abs((p1^2+c^2-a2^2-a3^2-d4^2)/sqrt(a^2+b^2))<=1)
   u3=acos((p1^2+c^2-a2^2-a3^2-d4^2)/sqrt(a^2+b^2))+atan(b/a);
else
   u3=acos((p1^2+c^2-a2^2-a3^2-d4^2)/abs(p1^2+c^2-a2^2-a3^2-d4
      ^2))-atan(b/a);
end

% 3. Step
% The rotation angle $u_2$ of e(2,1) is computed
u2=atan((c*(a2-a3*sin(u3)+d4*cos(u3))+p1*(a3*cos(u3)+d4*sin(u3)
   ))/(c*(a3*cos(u3)+d4*sin(u3))-p1*(a2-a3*sin(u3)+d4*cos(u3)))
   );

%Direct vector of e(4,1) and e(4,3), which are computed with
   Maple
```

```
e41=[-cos(u2+u3);-sin(u1)*sin(u2+u3);cos(u1)*sin(u2+u3)];
e411=round(e41*10);
e43=[sin(u2+u3);-sin(u1)*cos(u2+u3);cos(u1)*cos(u2+u3)];

 e61=[1 0 0;0cos(w2) -sin(w2);0sin(w2)cos(w2)]   *[cos(w1) 0
    sin(w1);0 1 0;-sin(w1) 0cos(w1)] *[1;0;0];
 e611=round(e61*10);

%4. step
%The rotation angle u4 is computed
if(norm(cross(e41,e61))==0)
   e51=[0;0;1];
else
   e51=(cross(e41,e61)/norm(cross(e41,e61)));
end
e511=round(10*e51);
uu4=cross(e43,e51);
uu41=norm(cross(e43,e51));
if(norm(cross(e43,e51))~=0)
  uu42=round(10*uu4/uu41);
 if(uu42(1)~=0)
    c0=uu42(1)/abs(uu42(1));
 else
    c0=0;
 end
 if(uu42(2)~=0)
    c1=uu42(2)/abs(uu42(2));
 else
    c1=0;
 end
 if(uu42(3)~=0)
    c2=uu42(3)/abs(uu42(3));
 else
    c2=0;
 end
 if(e411(1)~=0)
    c00=e411(1)/abs(e411(1));
 else
    c00=0;
 end
 if(e411(2)~=0)
    c11=e411(2)/abs(e411(2));
 else
    c11=0;
 end
 if(e411(3)~=0)
    c22=e411(3)/abs(e411(3));
```

```
 else
    c22=0;
 end

 if(c0==c00 && c1==c11 && c2==c22)
    u4=acos(e43'*e51);
 else
    u4=2*pi-acos(e43'*e51);
 end

elseif(e43(1)==e51(1)&&e43(2)==e51(2)&&e43(3)==e51(3))
    u4=0;
else
    u4=pi;
end

% 5. step
% The rotation angle u5 is computed
uu5=cross(e41,e61);
uu51=norm(cross(e41,e61));
if(uu51~=0)
    uu52=round(10*uu5/uu51);
 if(uu52(1)~=0)
    b0=uu52(1)/abs(uu52(1));
 else
    b0=0;
 end
 if(uu52(2)~=0)
    b1=uu52(2)/abs(uu52(2));
 else
    b1=0;
 end
 if(uu52(3)~=0)
    b2=uu52(3)/abs(uu52(3));
 else
    b2=0;
 end
 if(e511(1)~=0)
    b00=e511(1)/abs(e511(1));
 else
    b00=0;
 end
 if(e511(2)~=0)
    b11=e511(2)/abs(e511(2));
 else
    b11=0;
 end
```

```
 if(e511(3)~=0)
    b22=e511(3)/abs(e511(3));
 else
    b22=0;
 end
 if(b0==b00 && b1==b11 && b2==b22)
    u5=acos(e41'*e61);
 else
    u5=2*pi-acos(e41'*e61);
 end
elseif(e41(1)==e61(1)&&e41(2)==e61(2)&&e41(3)==e61(3))
    u5=0;
else
    u5=pi;
end

%6.step
%The rotation angle u6 is computed
%First of all, direct vector of e6,3 must be gave
e63=
[e61(1)^2*(1-cos(w3))+cos(w3) e61(1)*e61(2)*(1-cos(w3))-e61(3)*
   sin(w3) e61(1)*e61(3)*(1-cos(w3))+e61(2)*sin(w3);
e61(1)*e61(2)*(1-cos(w3))+e61(3)*sin(w3) e61(2)^2*(1-cos(w3))+
   cos(w3) e61(2)*e61(3)*(1-cos(w3))-e61(1)*sin(w3);
e61(1)*e61(3)*(1-cos(w3))-e61(2)*sin(w3) e61(2)*e61(3)*(1-cos(
   w3))+e61(1)*sin(w3) e61(3)^2*(1-cos(w3))+cos(w3)]*
[1 0 0;0cos(w2) -sin(w2);0sin(w2)cos(w2)]   *[cos(w1) 0sin(w1)
   ;0 1 0;-sin(w1) 0cos(w1)] *[0;0;1];


uu2=cross(e51,e63);
uu21=norm(cross(e51,e63));
if(norm(cross(e51,e63))~=0)
 uu22=round(10*uu2/uu21);
 if(uu22(1)~=0)
    a0=uu22(1)/abs(uu22(1));
 else
    a0=0;
 end
 if(uu22(2)~=0)
    a1=uu22(2)/abs(uu22(2));
 else
    a1=0;
 end
 if(uu22(3)~=0)
    a2=uu22(3)/abs(uu22(3));
 else
```

```
209     a2=0;
210  end
211  if(e611(1)~=0)
212     a00=e611(1)/abs(e611(1));
213  else
214     a00=0;
215  end
216  if(e611(2)~=0)
217     a11=e611(2)/abs(e611(2));
218  else
219     a11=0;
220  end
221  if(e611(3)~=0)
222     a22=e611(3)/abs(e611(3));
223  else
224     a22=0;
225  end
226
227  if(a0==a00 && a1==a11 && a2==a22)
228     u6=acos(e51'*e63);
229  else
230     u6=2*pi-acos(e51'*e63);
231  end
232 elseif(e51(1)==e63(1)&&e51(2)==e63(2)&&e51(3)==e63(3))
233     u6=0;
234 else
235     u6=pi;
236 end
```

B. Maple Code

Listing B.1:Calculation of $u_1 ... u_6$ from Denavit-Hartenberg parameters

```
1
2  > c1:=Matrix([[1,0,0,0],[d1,cos(a1),-sin(a1),0],[49.43,sin(a1),
     cos(a1),0],[p1,0,0,1]]);
3
4  > c2:=Matrix([[1,0,0,0],[d2,cos(a2),-sin(a2),0],[-p2,sin(a2),
     cos(a2),0],[0,0,0,1]]);
5
6  > c3:=Matrix([[1,0,0,0],[-d3,cos(a3),-sin(a3),0],[0,sin(a3),cos
     (a3),0],[-p3,0,0,1]]);
7
8  > c4:=Matrix([[1,0,0,0],[-d4,cos(a4),0,sin(a4)],[0,0,1,0],[p4,-
     sin(a4),0,cos(a4)]]);
9
10 > c5:=Matrix([[1,0,0,0],[d5,cos(a5),0,sin(a5)],[0,0,1,0],[p5,-
     sin(a5),0,cos(a5)]]);
11
12 > r1:=Matrix([[1,0,0,0],[0,1,0,0],[0,0,cos(u1),-sin(u1)],[0,0,
     sin(u1),cos(u1)]]);
13
14 > r2:=Matrix([[1,0,0,0],[0,1,0,0],[0,0,cos(u2),-sin(u2)],[0,0,
     sin(u2),cos(u2)]]);
15
16 > r3:=Matrix([[1,0,0,0],[0,1,0,0],[0,0,cos(u3),-sin(u3)],[0,0,
     sin(u3),cos(u3)]]);
17
18 > r4:=Matrix([[1,0,0,0],[0,1,0,0],[0,0,cos(u4),-sin(u4)],[0,0,
     sin(u4),cos(u4)]]);
19
20 > r5:=Matrix([[1,0,0,0],[0,1,0,0],[0,0,cos(u5),-sin(u5)],[0,0,
     sin(u5),cos(u5)]]);
21
22 > r6:=Matrix([[1,0,0,0],[0,1,0,0],[0,0,cos(u6),-sin(u6)],[0,0,
     sin(u6),cos(u6)]]);
23
24 > a1:=Pi/2;a2:=0;a3:=Pi/2;a4:=-Pi/2;a5:=Pi/2;
25 > p1:=804.8;p2:=1781;p3:=306.3;p4:=0;p5:=0;
26 > d1:=1012;d2:=0;d3:=156+49.43;d4:=1456;d5:=0;
27 > R:=r1.c1.r2.c2.r3.c3.r4.c4.r5.c5.r6;
```

```
28
29 > x6:=Vector([1,0,0,0]);
30
31 > R.x6;
32
33 > so1:=1012 +1781*cos(u2)-306.3*sin(u2+u3)  + 1456*cos(u2 + u3)
     ;
34
35 > s02:= -156*cos(u1)-sin(u1)*(804.8 - 1781*sin (u2) -306.3*cos(
     u2+u3)-1456*sin (u2 + u3));
36
37 > s03:= -156*sin(u1)+cos(u1)*(804.8 - 1781*sin (u2) -306.3*cos(
     u2+u3)- 1456*sin (u2 + u3));
38
39 > (s03+156*sin(u1))/cos(u1)=x;
40
41 > s02*cos(u1)=-156*cos(u1)^2-sin(u1)*(s03+156*sin(u1));
42
43 > s02*cos(u1)+s03*sin(u1)=-156;
44
45 > v1 := 1781*cos(u2)-306.3*sin(u2+u3)  + 1456*cos(u2 + u3);
46
47 > v2:=-1781*sin (u2) -306.3*cos(u2+u3)- 1456*sin (u2 + u3);
48
49 > v3:=v1*v1;
50
51 > v4:=v2*v2;
52
53 > v3+v4;
54
55 > v5:=combine(v3+v4 );
56
57 > d4^2+p2^2+p3^2;
58
59 > p2*2*d4;
60
61 > 2*p3*p2;
62
63 > pp:=1781*cos(u2)-306.3*sin(u2+u3)  + 1456*cos(u2 + u3);
64
65 > c:=-1781*sin (u2) -306.3*cos(u2+u3)- 1456*sin (u2 + u3);
66
67 > expand(pp);
68
69 > expand(c);
70
```

```
71 > pp2:=cos(u2)*(1781-306.3*sin(u3)+1456*cos(u3))-sin(u2)
      *(306.3*cos(u3)+1456*sin(u3));
72
73 > c2:=-sin(u2)*(1781-306.3*sin(u3)+1456*cos(u3))-cos(u2)
      *(306.3*cos(u3)+1456*sin(u3));
74
75 > sin(u2):=-(pp2-cos(u2)*(1781-306.3*sin(u3)+1456*cos(u3)))
      /(306.3*cos(u3)+1456*sin(u3));
76
77 > cos(u2):=(pp2+sin(u2)*(306.3*cos(u3)+1456*sin(u3)))
      /(1781-306.3*sin(u3)+1456*cos(u3));
78 >cos(u2):=-(c2*(306.3*cos(u3)+1456*sin(u3))-pp2*(1781-306.3*sin
      (u3)+1456*cos(u3)))/((1781-306.3*sin(u3)+1456*cos(u3))
      ^2-(306.3*cos(u3)+1456*sin(u3))^2);
79 >sin(u2):=-(c2*(1781-306.3*sin(u3)+1456*cos(u3))+pp2*(306.3*cos
      (u3)+1456*sin(u3)))/((1781-306.3*sin(u3)+1456*cos(u3))
      ^2-(306.3*cos(u3)+1456*sin(u3))^2);
```

List of Figures

Listings

Bibliography

[1]W. Aigner. *Visualization of time and time-oriented information:challenges and conceptual design*. PhD thesis, Vienna University of Technology, 2006.

[2]C. Bajaj. *Data visulization techniques*. Wiley, 1999.

[3]M. Bernardo. *Formal methods for the design of real-time system*. Springer, 2004.

[4]J.M. Van Verth / L.M. Bishop. *Essential Mathematics for Games and Interactive Applications*. Morgan Kaufmann, 2008.

[5]O. Bottema / B.Roth. *Theoretical Kinematics*. Dover Publications, New York, 1990.

[6]C. Chen. *Information visualization*. Springer, 2006.

[7]J.X. Chen. *Guide to Graphics Software Tools*. Springer, New York, 2003.

[8]J.W. Crenshaw. *Math Toolkit for Real-Time Programming*. CMP Books, USA, 2000.

[9]J.G de Jalon / E. Bayo. *Kinematic and Dynamic Simulation of Multibody Systems: The Real-Time Challenge*. Springer, 1993.

[10]P.J. Schneider / D.H. Eberly. *Geometric Tools For Computer Graphics*. Morgan Kaufmann Publishers, San Francisco, 2003.

[11]A. Gfrerrer. *Kinematik und Robotik*. Institut für Geometrie, Technical University of Graz, Graz, 2008.

[12]A. Gfrerrer. *LV Kinematik und Robotik, Übungsprojekt Ein serieller 6-Achs-Roboter mit Handgelenk*. Institut für Geometrie, Technical University of Graz, Graz, 2011.

[13]A.J. Hanson. *Visualizing Quaternions*. Morgan Kaufmann Publishers, San Francisco, 2005.

[14]L. He. *Human motion visualization in a virtual environment*. Computer Science Department, New Zealand, 2004.

[15]Mark / H. Hutchinson. *Real-Time Cameras: A Guide for Game Designers and Developers*. Morgan Kaufmann, 2005.

[16]ISO/IEC14772-1. *Information technology-Computer graphics and image processing-The Virtual Reality Modeling Language (VRML)-Part1:Functional specification and UTF-8 encoding*. 1997.

[17]A. Wirabhuanal / H.b. Haron / Jasril. *Industrial robot simulation software development using virtual reality modeling approach (VRML) and Matlab-Simulink toolbox*. Malaysia, 2005.

[18]O'. Joseph. *Computational Geometry In C*. Cambridge, 1997.

[19]K. Kasthurirangan. *Development of Manufacturing Systems Models Using VRML*. National Institute of Standards and Technology, Gaithersburg, MD., 1997.

[20]J.B. Kuipers. *Quaternions and Rotation Sequences*. Princeton University Press, New Jersey, 1999.

[21]S.H. Low. *Industrial Robotics Programming, Simulation and Applications*. plV pro literatur Verlag Robert Mayer-Scholz, 2007.

[22]The MathWorks. *Matlab R2009b Product Help*. http://www.mathworks.com.

[23]The MathWorks. *Robot Arm with Virtual Reality Scene*. Matlab 2009b/SimMechanics DEMOS.

[24]B. Siciliano / L. Sciavicco / L. Villani / G. Oriolo. *Robotics Modelling, Planning and Control*. Springer, 2008.

[25]P. Schrater. *Camera Parameters, Calibration and Radiometry*. 2005.

[26]R. Shumaker. *Virtual reality*. Springer, 2007.

[27]K. Takaya. *Multimedia Signals and Systems (Virtual Reality and VRML)*. University of Saskatchewan, 2008.

[28]E. Tittel. *Building VRML worlds*. McGraw-Hill, New York, 1997.

[29]M.W. Spong / S. Hutchinson / M. Vidyasagar. *Robot Modeling and Control*. Wiley, 2005.

[30]A. Yue / J. Hazen / L. Taylor / Y. Wen. The vrml educational manufacturing museum. http://www.umich.edu/~engr477/projectsf02/. vistited 01 Aug 2010.

[31]J. Hartman / J. Wernecke. *The VRML 2.0 handbook*. Addison-Wesley Professional, 1996.

[32]Wikipedia. http://www.wikipedia.org/.

[33]K. Wohlhart. *Dynamik*. Vieweg, Germany, 1997.